Université de Montréal


# IFT 6095 – Sujets en infographie – Exercice 1 : Rastérisation avec des *shaders*, et lancer de rayons sur CPU

Énoncé: 13/01/2014 Date de soumission: 26/01/2014

Le but du ce TP est de visualiser une scène très simple et de calculer l'illumination sur les surfaces avec deux algorithmes uniques: un moteur de rastérisation avec des *shaders*, et un lanceur de rayons. Vous utiliserez l'API de **WebGL** et **Javascript**, des technologies de web adaptées au prototypage rapide.

Je recommande soit **Chrome** ou **Firefox Beta (Aurora)** pour leurs excellents outils de développement de Javascript (et, dans le cas d'Aurora, de WebGL).

*Le plagiat est strictement interdit et sera traité très sérieusement. Si vous prévoyez utiliser du code pris en ligne je m'attends à ce que vous listiez la source **et** mettiez des commentaires pour bien illustrer votre compréhension; Sinon, je vous suggère fortement de lire le code, déchiffrer l'algorithme, et le réimplémenter vous-même.*

**Vous avez à soumettre** tout votre code source **HTML** and **JS** (propre, concis et bien documenté) pour vos démos de WebGL et Javascript, ainsi qu'un document court qui:

3. détaille les fonctionnalités clés de votre implémentation, y compris les détails algorithmiques, et
4. comprend *au moins une **(1)*** image pour chaque résultat/section/étape dans l'énoncé du TP.

Votre rapport devrait être écrit d'une manière concise: en-dehors des images, je m'attends à **3 pages au plus**.

## Javascript et WebGL

Vous allez implanter *deux **(2)*** pipelines de rendu simples:

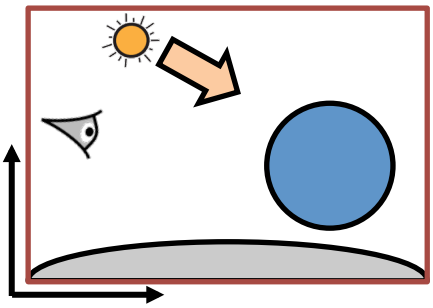
3. un démo **WebGL** (codé en **Javascript** et **GLSL**) qui **rastérise** la géometrie sur l'écran et qui utilise des ***shaders* simples** afin de transformer les sommets des objets et de calculer les couleurs des pixels,
4. un démo de lanceur de rayons en **Javascript (exclusivement)** pour générer une image d'une scène sans et avec des ombres simples.

Vous vous familiariserez avec le pipeline de rastérisation programmable et l'algorithme de *ray-tracing*. Avant de présenter les exigences du TP, je vous laisse avec quelques conseils:

- Je ne fournis aucun code de base substantiel.
- Lorsque vous codez les démos WebGL du TP, vous aurez l'option de coder à partir de zéro ou en utilisant une librarie de WebGL. Ici, le compromis est typique: les librairies exposent moins de détails du système et vous aideront à démarrer plus rapidement, mais vous pourriez manquer une opportunité de mieux comprendre les subtilités du pipeline. Partant de zéro, il faudra un peu plus de code ainsi qu'une connaissance plus développée du pipeline OpenGL/WebGL, mais cela peut être plus utile quand viendra temps de travailler sur le projet.
    - Si cela est votre premier cours en infographie, **utilisez `THREE.js` (**http://threejs.org/**)**: un API abstrait avec beaucoup de tutoriels et de documentation, tout en restant assez flexible.
    - Si vous voulez coder sans librairies, je suggère les liens suivants pour des renseignements supplémentaires: https://developer.mozilla.org/en-US/docs/Web/WebGL
    - Sinon, je peux recommender les *wrappers* suivants:
        - Lightgl.js : https://github.com/evanw/lightgl.js/
        - Litegl.js : http://tamats.com/webglstudio/litegl/
    - Dat.gui est une librairie de UI utile: http://workshop.chromeexperiments.com/examples/gui
- Vous êtes censés résoudre les problèmes techniques indépendamment avec les ressources à votre disposition (je serais surpris si vous avez besoin d'autre chose que les notes de cours et **Google**).
- N'ayez pas peur de coder! Je ne m'attends pas à plus de 400 lignes de code pour le TP, mais je ne veux pas vous décourager d'investiguer plusieurs solutions différentes: il y a certainement plusieurs solutions "correctes" et suffisamment de temps pour les implanter toutes. Vous serez notés presque exclusivement sur la réalisation des tâches exigées, mais je vais donner des points bonus pour certains styles de travaux supplémentaires (voir ci-dessous).
- Traitez ce TP comme un entraînement intensif sur l'affichage des pixels, et n'oubliez **de vous amuser**!

# 0 – Spécifications de la scène

Nous allons utiliser la même scène 3D pour chaque partie du TP. Voici les spécifications de la scène:



- Caméra perspective
  - position = (50,52,296)
  - direction (au lieu d'une position) *look-at* = |(0,-0.043,-1)|
  - vecteur de la direction *droite* = $\vec{x}$
  - champ de vision = 29.5 degrés
  - Lumière directionnelle à |(-10,4,10)|; intensité = $\pi$
- Scène avec 2 sphères diffuses (format = rayon, centre, $\rho$)
  - $10^5$, (80, $10^5$,0), (0.75, 0.75, 0.75)
  - 40, (50,40,0), (0.3,0.5,0.75)

Effectuez votre rendu sur un élément de *canvas* à résolution **512** × **384**.

# 1 – Rastérisation et *shaders* programmable [50 Points]

Livrable #1 **[35 points]**: Codez un démo WebGL pour créer une image selon les spécifications de la scène de la Section 0, et puis codez des *shaders* de sommets (*vertices*) et de pixels pour effectuer les opérations suivantes:

- Le ***Vertex Shader*** affectera la transformation *modèle-vue-projection* du pipeline d'OpenGL, en transformant les sommets de la scène (spécifiés dans le code Javascript) en espace de projection. Les normales seront également (et "proprement") transformées et transmises au *pixel shader*. Après cet étape du pipeline, le GPU va automatiquement *rastériser* les formes géométriques en appliquant la division perspective (et possiblement un test de profondeur avec le *z-buffer*, selon les paramètres WebGL). Vous aurez accès aux matrices de transformation *modèle*, *vue* et *projection*; parfois, ces matrices sont (pré-)combinées ou pas dans l'API et renduess disponible dans le *shader*.
- Le ***Pixel Shader*** utilisera les propriétés de la lumière, les normales et les reflectances des objets afin de calculer une couleur finale au pixel selon l'équation d'illumination directe pour lumière directionnelle:

$$L_o(n) = I\,\frac{\rho}{\pi}\max(n\cdot L, 0)$$

où $I$ est l'intensité de la lumière, $\rho$ est la réflectance de l'object, $n$ est la normale, et $L$ la direction (normalisée) de la lumière. Soyez prudents d'utiliser le même système de coordonnées pour chacun des termes (je suggère l'espace monde) avant d'appliquer cette équation.

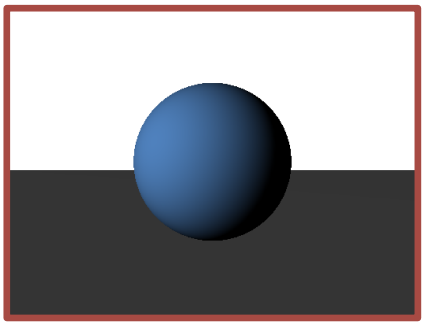


**Figure 1** – Résultat prévu pour le livrable #1.

**Remarque:**

- Si vous utilisez **`THREE.js`**, je m'attends à ce que vous ne vous basiez pas sur les implémentations du modèle d'illumination disponible dans la librairie (notamment, **`THREE.MeshLambertMaterial`**); vous avez besoin de coder le modèle vous-même (par exemple, avec **`THREE.ShaderMaterial`**).
- Si vous avez plus de 60 lignes de code dans vos *shaders*, vous faites quelque chose de mal.

Livrable #2 **[15 points]**: Chaque propriété d'un sommet émis du *vertex shader* (et alors, disponible en tant qu'entrée au *pixel shader*) sera potentiellement interpolée le long de la face du triangle/polygone pendant la rastérisation. Vous pouvez activer ou désactiver ce comportement (voir **`GL_SMOOTH`**) avec un appel WebGL (avec Javascript). Cela peut causer des artéfacts visibles, tout dépendant de la densité des triangles dans la scène

(voir les différences entre les ***shadings*** de *Phong* et *Gouraud*). Modifiez le taux de tessellation dans votre scène afin d'illustrer cet artéfact. Comparez des versions à faible tessellation et à haute tessellation, toutes les deux avec et sans l'interpolation de *Gouraud*, et **discutez de vos observations brièvement**.

# 2 – Lancer de rayons avec Javascript sur le CPU [50 Points]

Livrable #3 **[30 points]**: Codez un démo en Javascript qui trace des rayons de la caméra à travers chaque pixel (voir les spécifications de la caméra – Section 0) sur la même scène, et qui émet une couleur (selon l'équation du Livrable #1) sur la surface la plus proche. Le résultat correspondra effectivement à l'image dans la **Figure 1**.

Livrable #4 **[20 points]**: Utilisez un seul rayon d'ombre pour calculer des ombres dures venant de la lumière directionnelle. Les régions en ombre seront complètement noires.

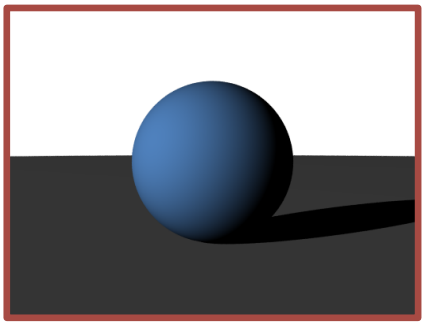


**Figure 2** – Résultat prévu pour le livrable #4.

# 3 – Bonus [jusqu'à 25 Points]

Impressionnez-moi. Cela demande généralement d'implanter de nouvelles fonctionalités, que ce soit un modèle de réflectance novateur, un effet de distribution comme le flou de mouvement ou des réflexions floues, l'éclairage environnemental, ou quelque chose de plus intéressant. Rappelez-vous que les résultats les plus convaincants en rendu combinent une capacité d'ingéniosité de 90% avec une capacité artistique de 10%.

Je n'accorde **aucun** point pour des poursuites non significatives, par exemple: l'ajout de deux sphères dans la scène, l'utilisation d'une fonctionalité déjà implantée dans la librairie utilisée, le changement de la couleur de la lumière, etc.

Si votre idée supplémentaire générera une image *cool* et si elle exige un effort de développement théorique et/ou technique en-dehors des exigences du TP, ça s'approche de quelque chose d'intéressant. J'ai hâte de vous fournir des commentaires sur les idées que vous pensez réaliser pour le bonus.

## Quelques images motivantes

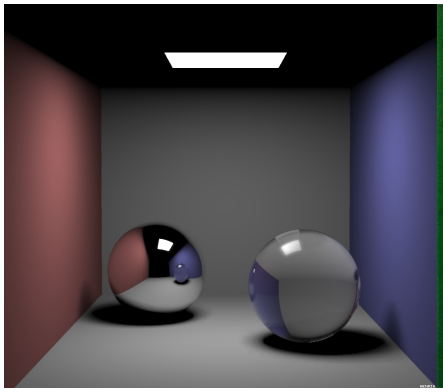


**Figure 3** – Ombres étendues, réflexion miroir et réfraction dan la scène de la Cornell Box [Jensen99].

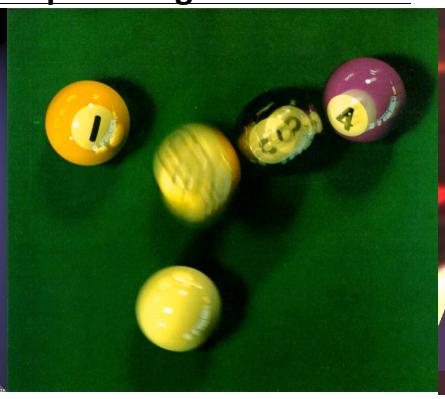


**Figure 4** – Une table de billard avec un flou de mouvement et de réflexions [Cook84].

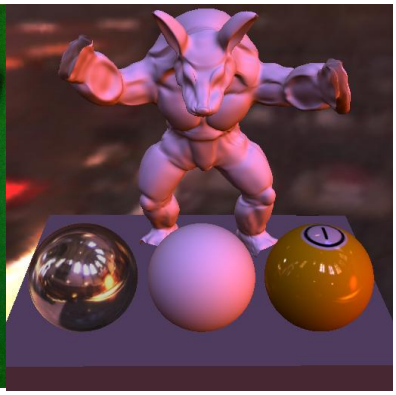


**Figure 5** – *Irradiance environment mapping* avec des harmoniques sphériques [RH01].