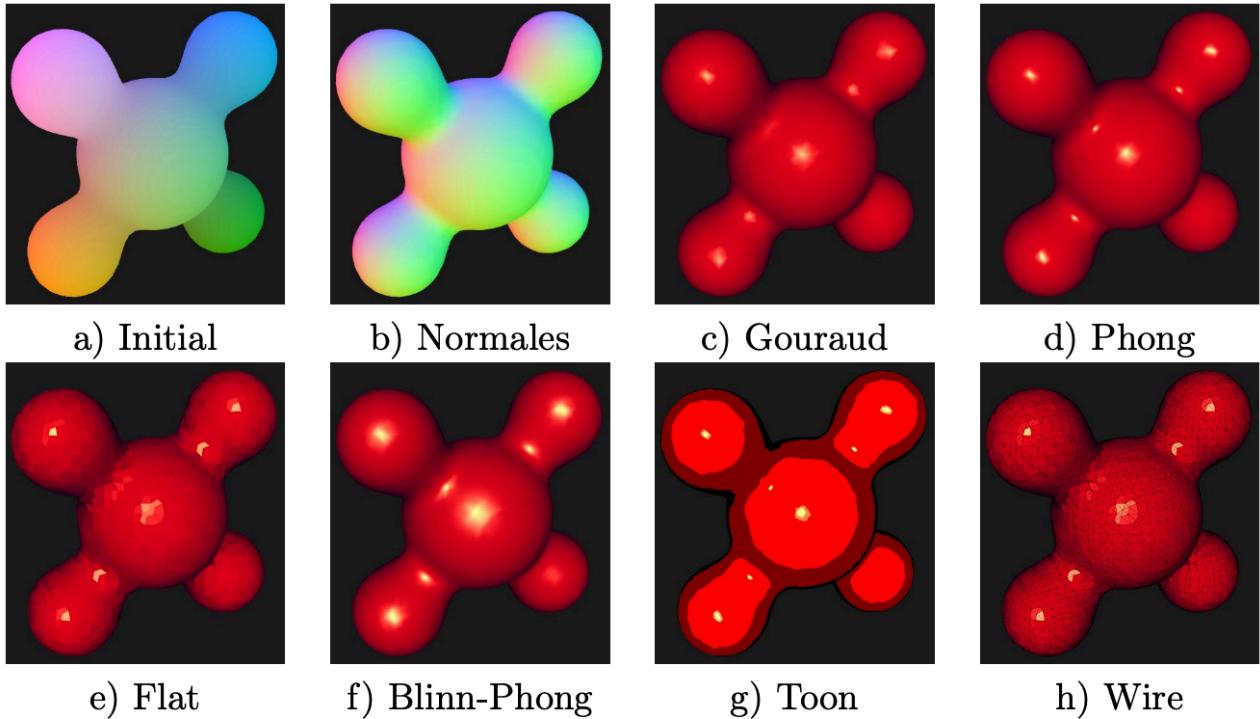


TP4 - Modèles d'illumination

Amad Salmon | Synthèse d'Image - Polytech Grenoble, INFO4, 2020-2021



TP4 - Modèles d'illumination

Choix d'implémentation

Choix du modèle d'illumination

Nom des variables

Shading de Gouraud

Shading de Phong

Flat shading

Choix d'implémentation

Choix du modèle d'illumination

Afin de faciliter le passage entre les deux modèles d'illumination et afin d'éviter de devoir changer manuellement le nom des shaders à charger selon le modèle voulu, nous mettons en place une variable `choix_shader` qui peut prendre comme valeur les constantes `SHADING_DE_GOURAUD`, `SHADING_DE_PHONG`, `SHADING_DE_PHONG_FLAT`. Selon `choix_shader`, le programme `main.cpp` charge le shader program voulu.

Par exemple, pour choisir d'illuminer le dessin avec un shader de Phong AVEC interpolation des normales, il suffit, (*ligne 96 du `main.cpp`*), d'écrire `int choix_shader = SHADING_DE_GOURAUD;`.

Nom des variables

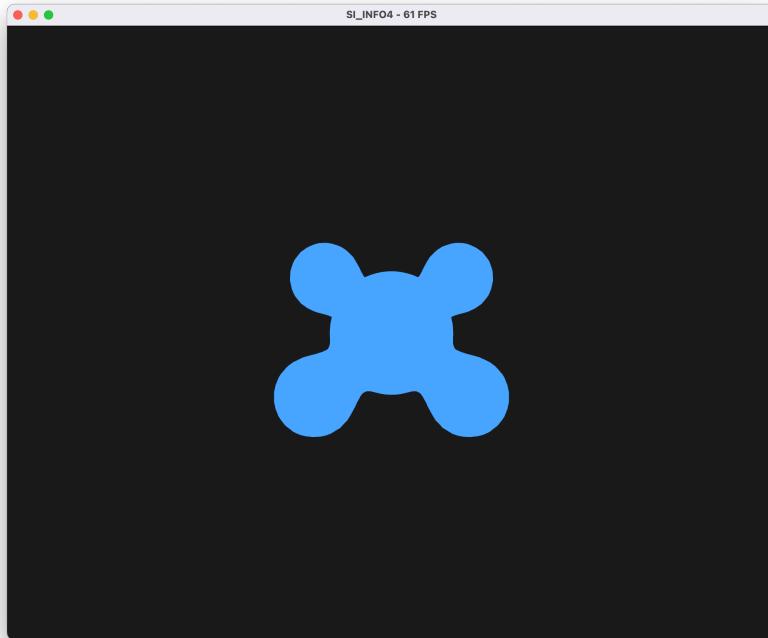
Variable mathématique	Nom dans le code	Fonction
L_f	<code>L_f</code>	Couleur résultante calculée par le modèle
ρ_a	<code>rho_a</code>	Coefficient d'atténuation associé à la composante ambiante
ρ_d	<code>rho_d</code>	Coefficient d'atténuation associé à la composante diffuse
ρ_s	<code>rho_s</code>	Coefficient d'atténuation associé à la composante spéculaire
L_a	<code>L_a</code>	Couleur de la composante ambiante
L_d	<code>L_d</code>	Couleur de la composante diffuse
L_s	<code>L_s</code>	Couleur de la composante spéculaire
n	<code>n</code>	Vecteur normal de la surface au point p
l	<code>l</code>	Vecteur de norme dans la direction de la lumière
r	<code>r</code>	Vecteur réflexion de l par rapport à n : $r = l - 2 < n, l > n$
e	<code>e</code>	Vecteur de norme 1 dans la direction entre le point p de la surface et la position c de l'oeil
s	<code>s</code>	Brillance (<i>shininess</i>)

Shading de Gouraud

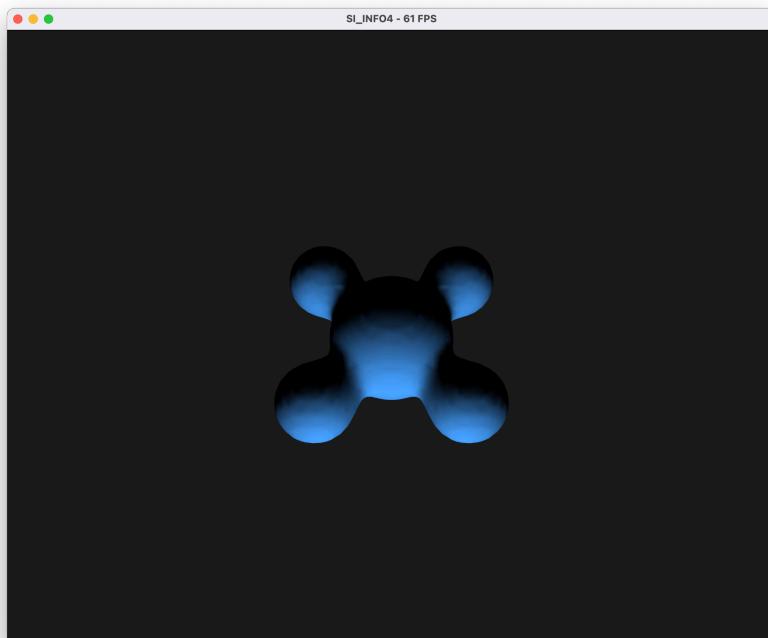
On implémente le shading de Gouraud grâce aux shaders `vertex_gouraud.glsl` et `fragment_gouraud.glsl`.

Mettons $\rho_a = \rho_d = \rho_s = 1.0$, $L_a = L_d = L_s$ (*couleur bleutée*), et une brillance $s = 1.0$.

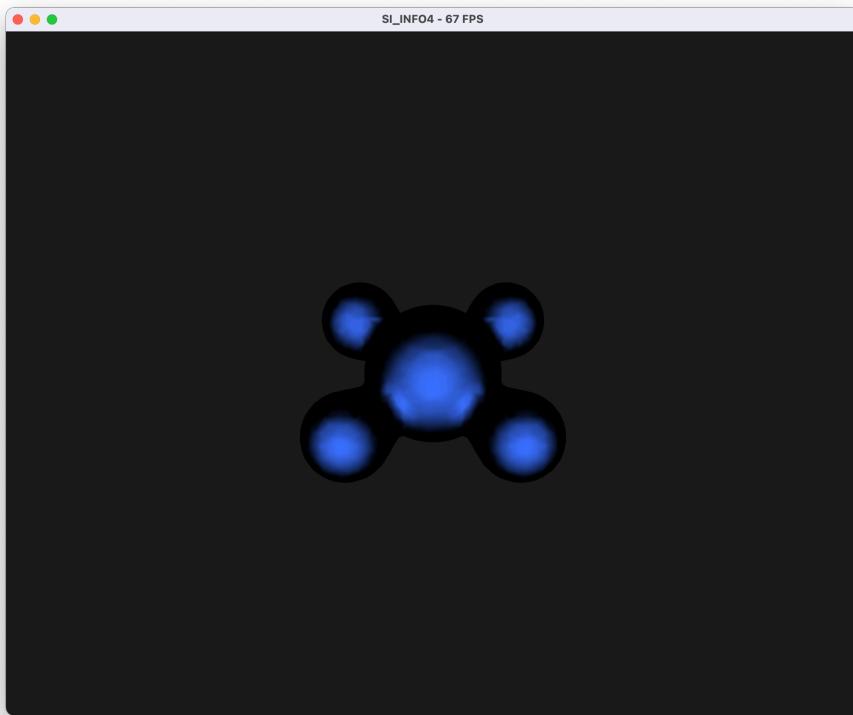
On affiche la composante ambiante avec `L_f = ambiant = rho_a * L_a`. Le résultat est :



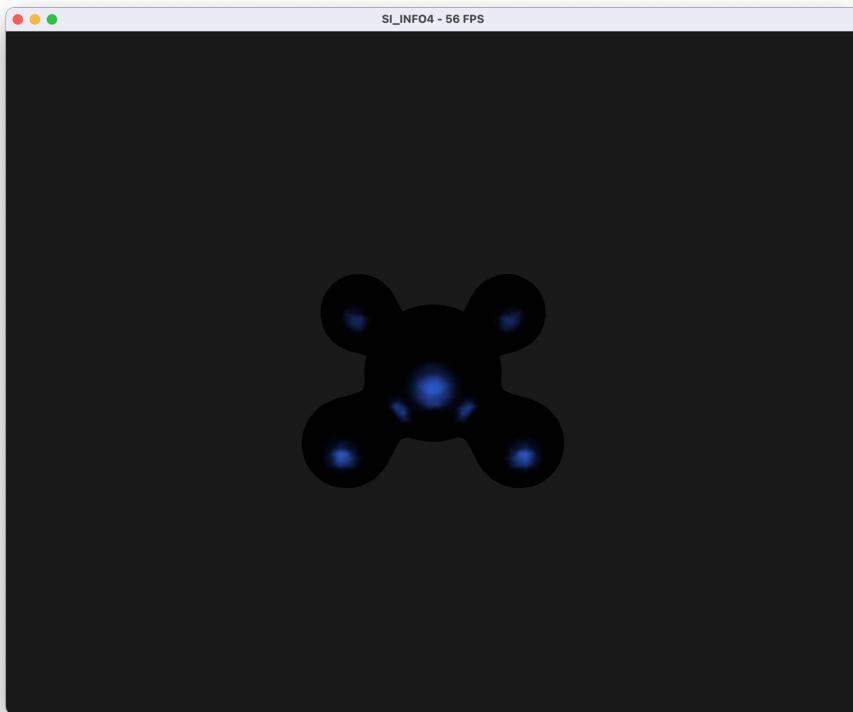
On affiche la composante diffuse avec `L_f = diffus = rho_d * L_d * max(-dot(n, l), 0.0)`. Le résultat est :



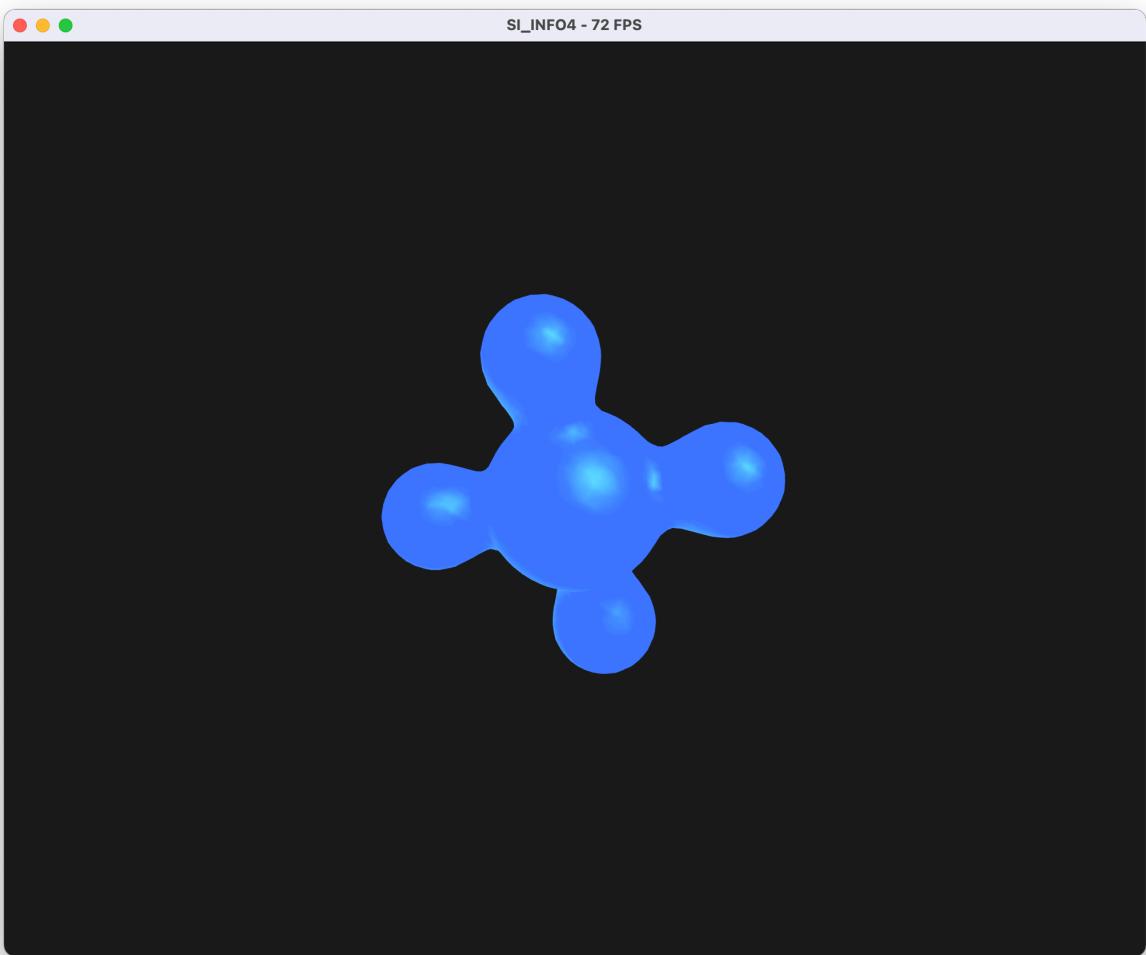
On affiche la composante spéculaire avec `L_f = speculaire = rho_s * L_s * pow(max(-dot(r, e), 0.0), s)`. Le résultat est :



En augmentant la brillance à `s = 10.0`, on obtient des tâches de spécularité plus définies :



En additionnant les trois composantes avec `L_f = ambiant + diffus + speculaire`, on obtient un objet dont le modèle d'illumination suit le module de Gouraud :



Shading de Phong

On implémente le shading de Phong grâce aux shaders `vertex_phong.glsl` et `fragment_phong.glsl`.

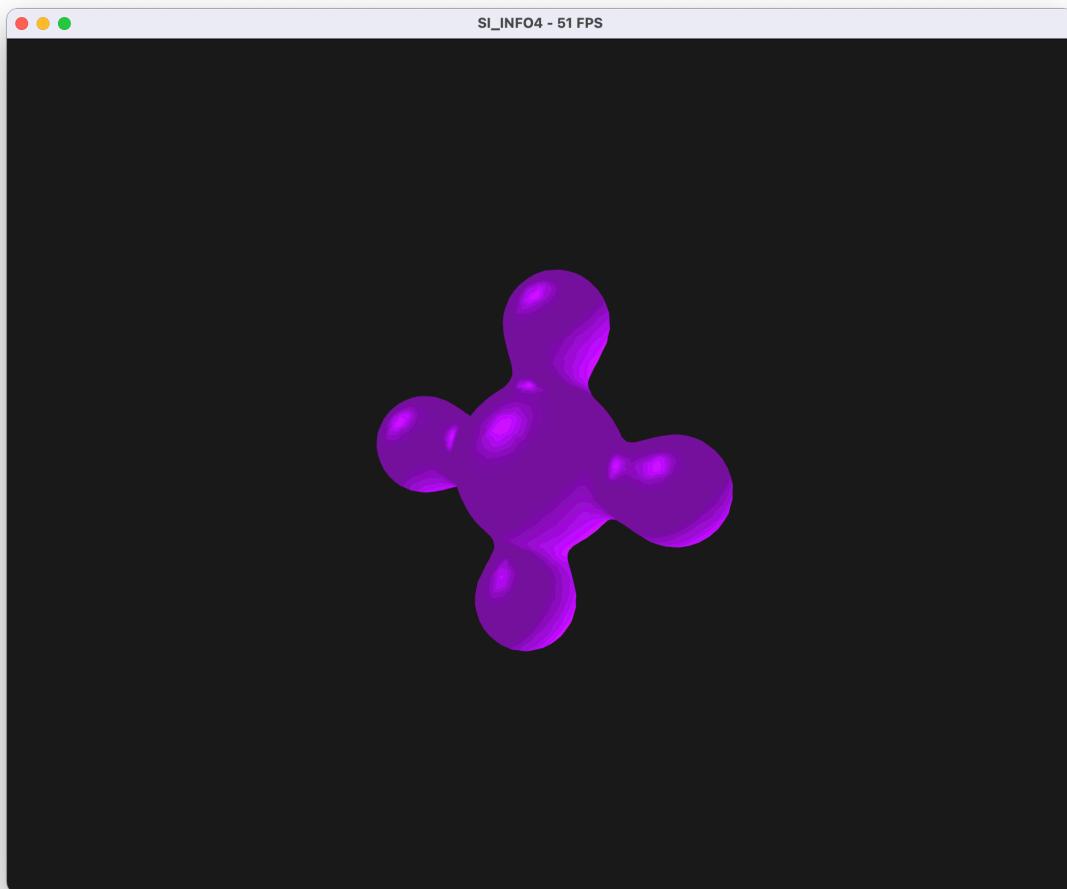
Dans le `vertex_phong.glsl`, on calcule et fait sortir les variables suivantes :

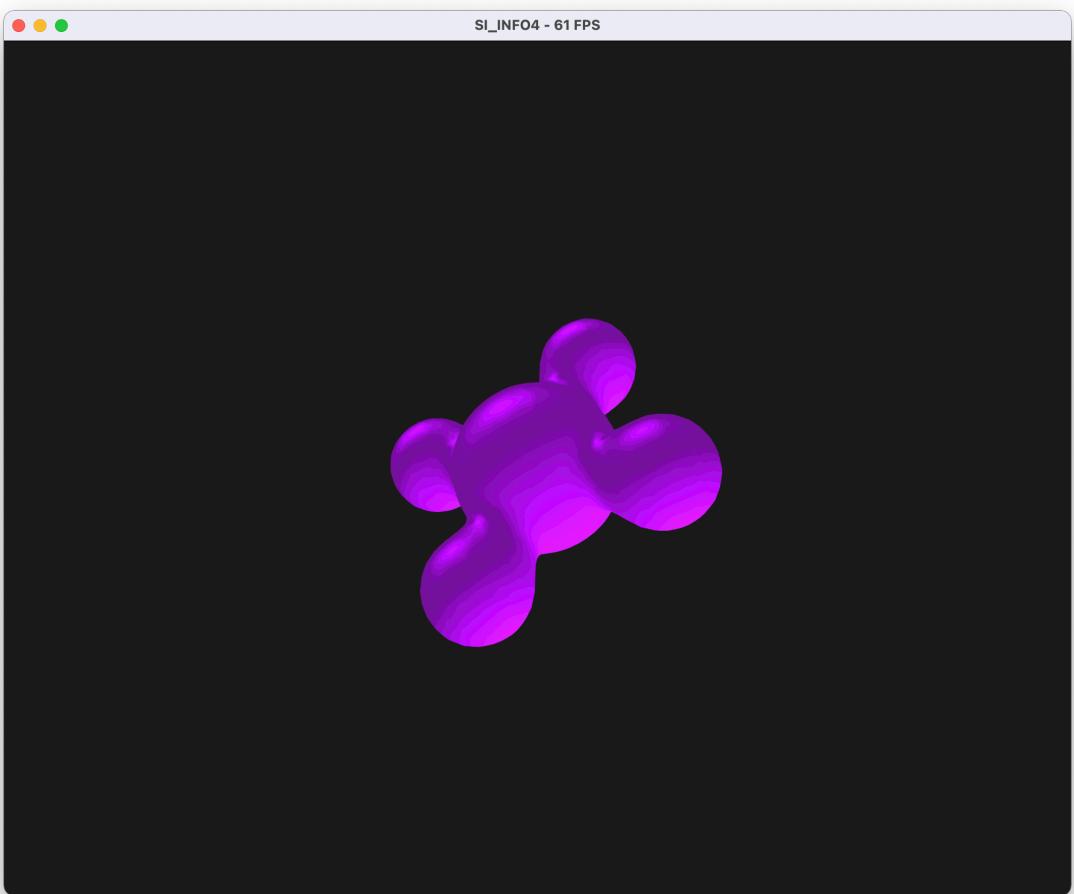
```
/* -- Données en sortie -- */
out vec4 posInWorldSpace;           // point p (position) dans le repère monde
out vec4 normalInWorldSpace;        // normales dans le repère monde
out vec4 cameraPosInWorldSpace;     // caméra dans le repère monde
```

Et on les fait entrer dans le `fragment_phong.glsl` :

```
/* -- Données en entrée -- */
in vec4 posInWorldSpace;
in vec4 normalInWorldSpace;
in vec4 cameraPosInWorldSpace;
```

Pour différencier l'objet dessiné lorsqu'il est illuminé par le shading de Gouraud de lorsqu'il l'est par le shading de Phong, ici le shading de Gouraud donne une couleur violette à l'objet. Le résultat est le beau blob suivant :





Flat shading

On empêche OpenGL d'interpoler les normales dans le shading de Phong en placant le mot-clé `flat` devant la déclaration de l'attribut concerné, c'est à dire devant le `out` dans le vertex shader et devant le `in` dans le fragment shader.

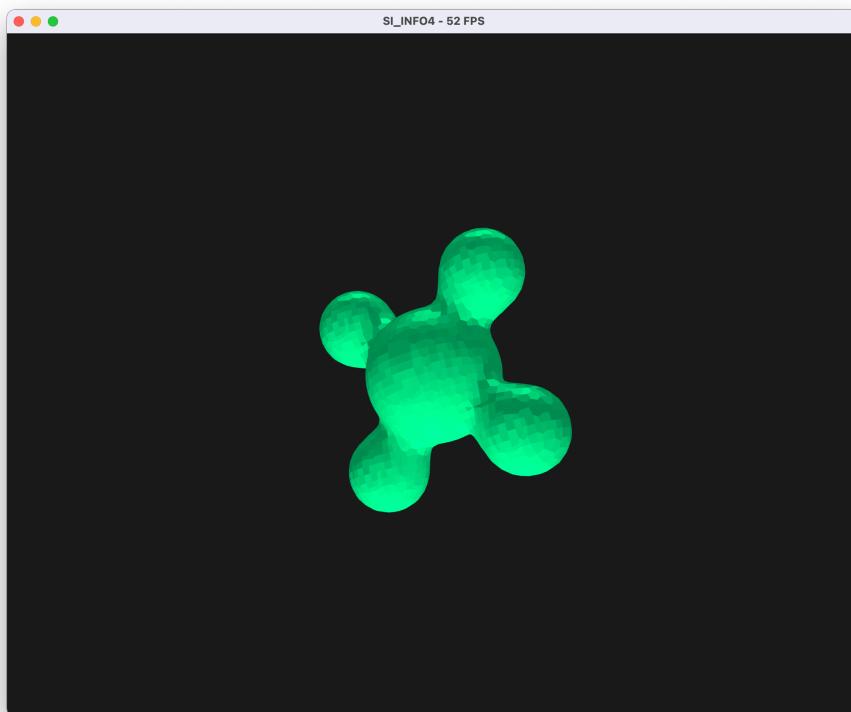
Ainsi, dans `vertex_flat_phong.glsl` :

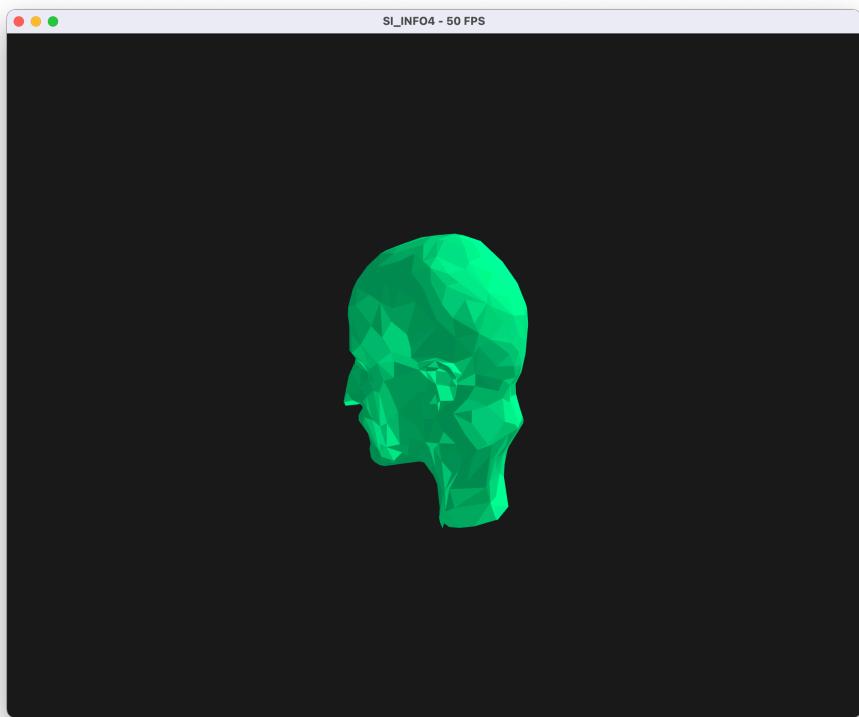
```
/* -- Données en sortie -- */
flat out vec4 posInWorldSpace;           // point p (position) dans le repère monde
flat out vec4 normalInWorldSpace;         // normales dans le repère monde
flat out vec4 cameraPosInWorldSpace;      // caméra dans le repère monde
```

Et dans le `fragment_flat_phong.glsl` :

```
/* -- Données en entrée -- */
flat in vec4 posInWorldSpace;
flat in vec4 normalInWorldSpace;
flat in vec4 cameraPosInWorldSpace;
```

Afin de différencier ce modèle des autres, les objets qui en seront peints le seront d'une couleur verte. Dans le main, ligne 97, on indique `int choix_shader = SHADING_DE_PHONG_FLAT;`. Alors, le résultat est le suivant :





On voit effectivement que les surfaces ne sont plus lisses comme avant, mais plutôt représentées par des triangles plats aux arêtes bien définies.