

Rapport du projet de Synthèse d'image

Clery Arsène & Kramer Axel

Sommaire :

1.....	Introductio
2.....	Modélisatio n de pikachu
3.....	Gestion et application des textures
4.....	Animations
5.....	Lumières/éclairages
6.....	Architecture du projet

1. Introduction

Le projet de Synthèse d'Image réalisé dans le cadre de la Licence 3 avait pour objectif de modéliser, texturer, éclairer et animer un personnage en 3D : Pikachu. L'ensemble du projet a été développé en C++ avec OpenGL, en respectant les contraintes imposées dans le sujet du

projet, notamment l'utilisation de primitives paramétriques personnalisées (dans notre cas nous en auront qu'une et nous en reparlerons plus tard), la gestion manuelle des textures et l'intégration de plusieurs types de lumières.

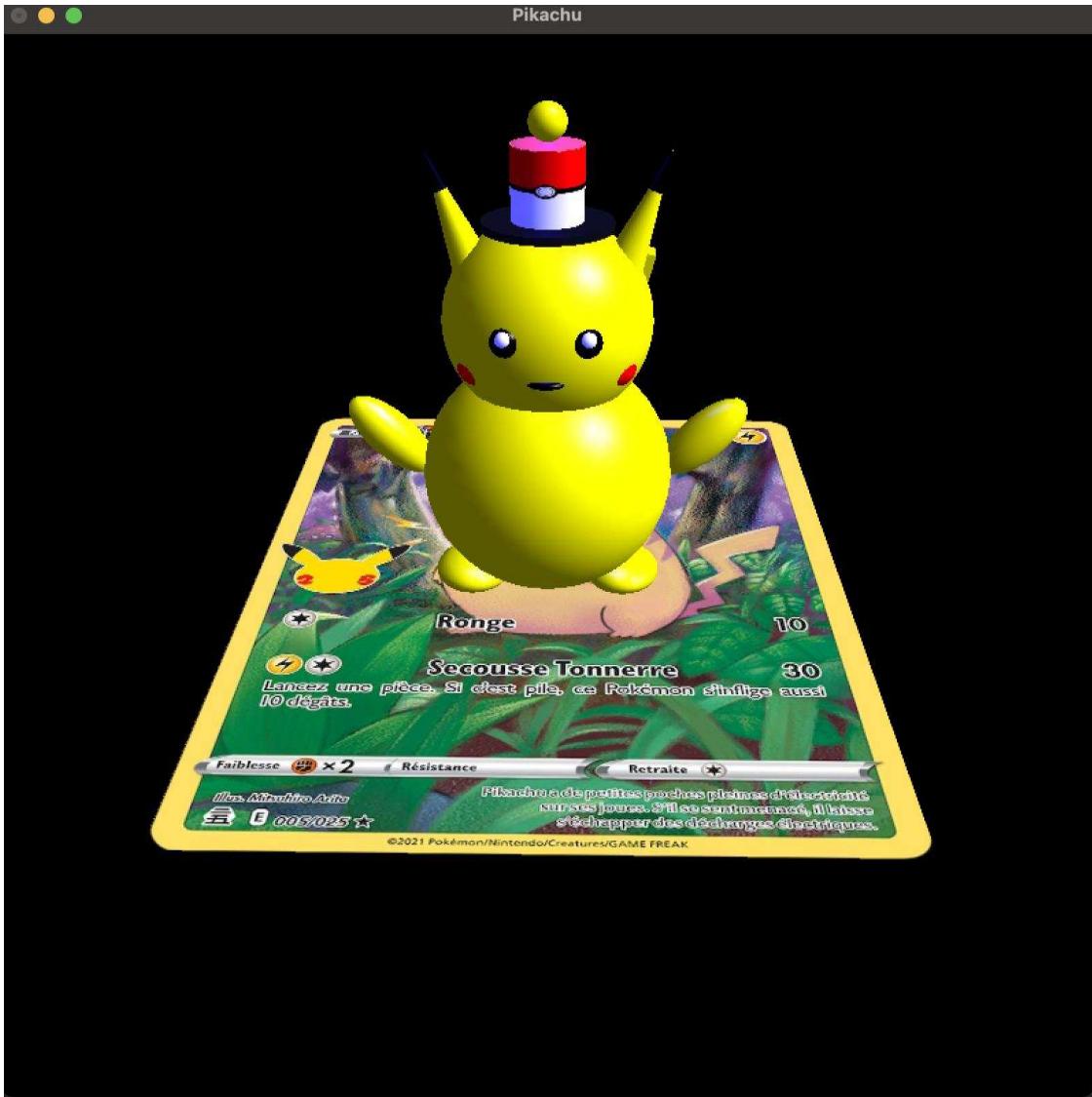


qui étaient interdites, il s'agissait ici de produire notre propre description mathématique d'une surface et d'en dériver un maillage exploitable par OpenGL. Nous voulions faire la casquette à hélice du pikachu par une demi-sphère paramétrique. Nous avons finalement abandonnée l'idée de la casquette pour le remplacer par un haut de forme et vite opté pour une représentation relativement "simple" de pikachu pour pousser à fond et être sûrs de pouvoir assurer toute la partie plus "compliquée" (animations et textures).

Notre projet a débuté par une phase d'analyse et de recherche sur les formes et proportions du personnage. Nous avons pris un modèle de pikachu (à gauche) sur lequel nous allons ensuite baser tout notre code et notre modélisation. Notre idée de base était de le modéliser entièrement et de faire des animations en rapport avec ce pikachu "déTECTIVE".

L'un des points essentiels du projet était l'implémentation d'au moins une primitive générée à partir d'une représentation paramétrique. Contrairement aux quadriques

Voilà un bref aperçu de notre pikachu :



Une fois le modèle de base construit, l'étape suivante a été de faire l'animation idle (qui s'exécute tout le temps), comparable à notre respiration humaine, léger mouvement du diaphragme. Nous avons opté pour un balancement des bras, des oreilles et de la queue du pikachu. Cela permet d'utiliser plusieurs types de rotation.

Le réalisme du modèle a ensuite été renforcé par l'intégration de deux types de lumières. Une lumière directionnelle, inspirée d'un éclairage solaire, permet de créer des ombres douces et homogènes. À cela s'ajoute une lumière ponctuelle positionnée près du personnage, renforçant les volumes et les contrastes. L'ajustement des composantes ambient, diffuse et specular nous ont permis d'obtenir un aspect visuel plaisant, tout en facilitant la lecture du modèle sous différents angles.

Afin d'offrir une interaction fluide et intuitive avec la scène, nous avons développé un système de caméra contrôlable via les touches du clavier. Les touches 'z' et 'Z' permettent de zoomer ou dézoomer, tandis que les flèches contrôlent la rotation autour du personnage.

Ce rapport revient en détail sur chaque étape de développement et le résultat final est un Pikachu entièrement modélisé, texturé, éclairé et animé, qui respecte les consignes du sujet.

2. Modélisation de Pikachu

La modélisation de Pikachu est l'étape centrale du projet, puisqu'elle est à la base de tout le travail réalisé ensuite sur les textures, les animations, l'éclairage et la caméra. Notre objectif était de concevoir un pikachu reconnaissable, et suffisamment structuré pour permettre des animations convaincantes tout en respectant les contraintes du sujet : utiliser majoritairement des primitives géométriques simples (sphères, cylindres, cônes, etc.) et intégrer au moins une primitive paramétrique personnalisée.

2.1. Référence et choix du style

Comme expliqué dans l'introduction, nous avons commencé par étudier un modèle de Pikachu version « détective », que nous avions choisi comme modèle. L'idée était de reproduire au mieux cette version, éventuellement en incluant une casquette à hélice reposant sur une demi-sphère paramétrique.

Cependant, ce design s'est rapidement révélé trop ambitieux pour la durée du projet, car il aurait nécessité une modélisation plus fine (notamment une texture sur une demi-sphère). Avant même d'essayer, nous sommes partis sur le cylindre comme forme paramétrique déjà fait en cours en grande partie. Par conséquent nous avons dû changer notre idée autour de notre casquette à hélice et toutes les animations qui en découlaient.

2.2. Décomposition du personnage en primitives

La méthode utilisée pour modéliser Pikachu s'appuie sur une approche en plusieurs blocs. Le personnage est découpé en plusieurs blocs indépendants, chacun représenté par une ou plusieurs primitives (non paramétrique). Ce découpage facilite les animations ultérieures, puisqu'il suffit ensuite d'appliquer des transformations locales sur chaque partie.

Voici la structure principale retenue :

- **Tête** : sphère , pour rappeler le visage arrondi de Pikachu.

- **Corps** : sphère, on aurait pu l'étirer non uniformément mais nous avons décidé de la laisser comme cela. Après plusieurs tests, cela nous permet de reconnaître plus facilement pikachu.
- **Bras** : Sphères "scalées" /déformées non homogènement sur les trois composantes et qui ont subi une rotation.
- **Pieds**: deux ellipsoïdes (sphères déformées) écrasés à leur base, positionnés pour assurer la stabilité visuelle.
- **Oreilles** : 2 cônes par oreille, un est noir pour l'extrémité de l'oreille
- **Queue** : primitive faite de plusieurs cubes déformés, donnant la forme caractéristique d'éclair. Tout en jouant avec les couleurs.
- **Joues** : Petites sphères collées sur la surface du visage.
- **Haut-de-forme** : 2 cylindres en paramétrique, un à la base, l'autre est le corps du chapeau.

Cette structure nous a permis de créer un modèle fidèle aux proportions globales du personnage.

2.3. La primitive paramétrique personnalisée

Le sujet imposait la création d'une primitive paramétrique générée à partir de sa représentation mathématique. Nous avions d'abord prévu d'utiliser cette primitive pour modéliser une demi-sphère (la casquette à hélice du Pikachu détective). L'idée, étant écartée, nous avons choisi le cylindre en primitive paramétrique.

Cette primitive n'est pas seulement un "tube" : il s'agit d'un volume complet comprenant :

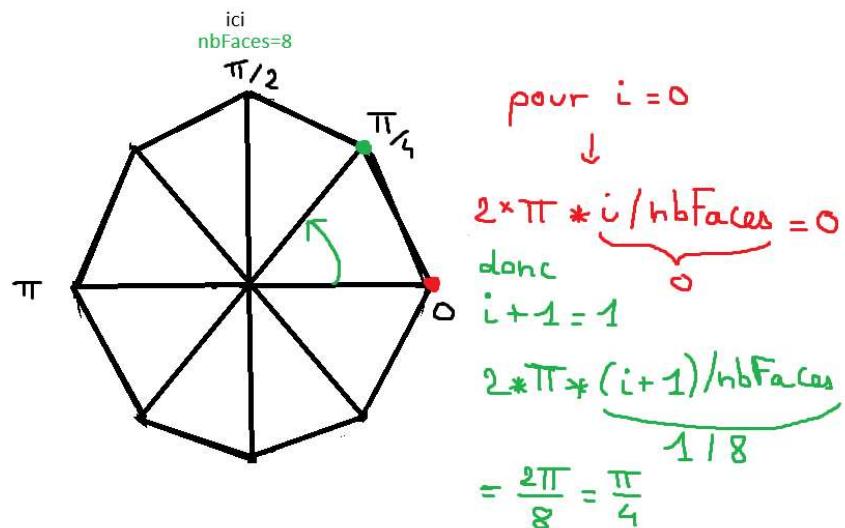
- les deux disques(faces supérieure et inférieure),
- la surface latérale,
- les normales calculées à la main,
- les coordonnées de texture générées manuellement,
- et un contrôle précis du nombre de faces pour ajuster la qualité du maillage.

La génération du cylindre repose sur un paramétrage angulaire :

on découpe le cercle en nbFaces segments, ce qui nous permet de produire un polygone régulier approximant la surface.

Chaque face latérale est obtenue à partir de deux angles successifs :

- $\Theta_0 = 2.0 * M_PI * i / nbFaces;$
- $\Theta_1 = 2.0 * M_PI * (i+1) / nbFaces;$



Pour chaque angle, on calcule les coordonnées du point sur le cercle :

$$x = \text{posX} + r \cos(\theta), \quad z = \text{posZ} + r \sin(\theta)$$

Puis le code crée le rectangle grâce aux 4 points maintenant connus.

Génération des deux disques:

Les deux faces du cylindre ne sont pas générées automatiquement :

elles sont reconstruites point par point.

Le code génère :

- un tableau de points régulièrement espacés,
- un centre du disque,

- puis triangule la surface en *nbFaces* triangles.

```
tabPointDisque[i].x = posX + rayon * cos((2 * i * M_PI) / nbFaces);
tabPointDisque[i].z = posZ + rayon * sin((2 * i * M_PI) / nbFaces);
tabPointDisque[i].y = posY + hauteur;
```

3.Gestion et application des textures

La gestion des textures constitue une partie essentielle de notre projet, car elle permet d'améliorer fortement le rendu visuel de Pikachu et de satisfaire une contrainte importante du sujet : utiliser au moins deux textures, dont une plaquée sur une surface et une autre enroulée autour d'une primitive paramétrique ;

Dans notre projet, toutes les coordonnées de texture ont été calculées manuellement, conformément aux consignes.

3.1. Chargement manuel des textures (JPEG)

Les textures utilisées dans notre scène sont des images JPEG chargées en mémoire grâce à la bibliothèque libjpeg.

Le chargement se fait dans la fonction :

```
loadJpegImage("carte.jpg", image, largimg , hautimg);
loadJpegImage("pokeball.jpg", image2, largimg2,hautimg2);
```

3.2. Création des textures OpenGL

Une fois les pixels chargés, les textures sont envoyées à la carte graphique :

```
glBindTexture(GL_TEXTURE_2D, 1);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, largimg, hautimg, 0,
GL_RGB, GL_UNSIGNED_BYTE, image);
```

La texture est configurée avec :

- `GL_LINEAR` pour le filtrage (meilleure qualité)
- `GL_MODULATE` pour combiner la lumière et la texture

Nous utilisons deux textures différentes :

`Carte.jpg`, la carte sur laquelle est le pikachu.

`Pokeball.jpg`, la texture sur le chapeau.

3.3. carte.jpg

La première texture est appliquée sur le sol sous Pikachu, qui est un simple quadrilatère.

Dans affichage() :

```
glEnable(GL_TEXTURE_2D);
 glBindTexture(GL_TEXTURE_2D, 1);

 glBegin(GL_QUADS);
 glTexCoord2f(0, 1); glVertex3f(-1, 0, 1);
 glTexCoord2f(0, 0); glVertex3f(-1, 0, -1);
 glTexCoord2f(1, 0); glVertex3f( 1, 0, -1);
 glTexCoord2f(1, 1); glVertex3f( 1, 0, 1);
 glEnd();
```

Ici :

- Les coordonnées de texture (u,v) sont définies à la main.
- La texture est simplement plaquée sur la surface.

Cela remplit la première exigence : une texture plaquée sur une face.

3.4. pokeball.jpg

La seconde texture (`pokeball.jpg`) est utilisée sur une primitive paramétrique de notre modèle : **le cylindre**.

Dans la classe Cylindre, les coordonnées de texture sont calculées manuellement en fonction du paramètre angulaire :

```
float u0 = (float)i / nbFaces;  
float u1 = (float)(i + 1) / nbFaces;
```

Ce choix permet d'enrouler la texture autour du cylindre comme ceci :

u=0 -----> u=1
| |
| (texture) |
| |
v=1 v=1

La dimension **u** correspond à l'angle autour du cylindre,

tandis que **v** correspond à la hauteur :

```
glTexCoord2f(u0, 1.0f); glVertex3f(x0, yBas, z0);  
glTexCoord2f(u1, 1.0f); glVertex3f(x1, yBas, z1);  
glTexCoord2f(u1, 0.0f); glVertex3f(x1, yHaut, z1);  
glTexCoord2f(u0, 0.0f); glVertex3f(x0, yHaut, z0);  
.
```

La texture suit la surface paramétrique du cylindre sans coupure

4. Animations

Il fallait d'après le sujet, faire deux types d'animation.

- Une animation permanente

- Une animation se lançant à l'appui spécifique sur une touche

4.1. Animation permanente

```
void Tortue::IdleAnimation() {
    frameAnim += 0.1;

    if (!is_started) return;

    if (phase == 0) {
        espanim += 2;
        poke=espanim;
        if (espanim >= 90) {
            espanim = 90;
            phase = 1;
        }
    } else if (phase == 1) {
        espanim -= 2;
        if (espanim>39)
            poke=espanim;
        if (espanim <= 0) {
            espanim = 0;
            phase = 0;
            is_started = 0;
        }
    }
}
```

Voici la fonction qui concerne les deux animations (bien qu'elle soit nommée par le nom de l'animation Idle). Nous allons nous intéresser dans notre cas à la première ligne. `frameAnim+=0.1;` Ici, à chaque fois que la fonction est appelée (tout le temps) grâce à la fonction `idle()` dans le main.

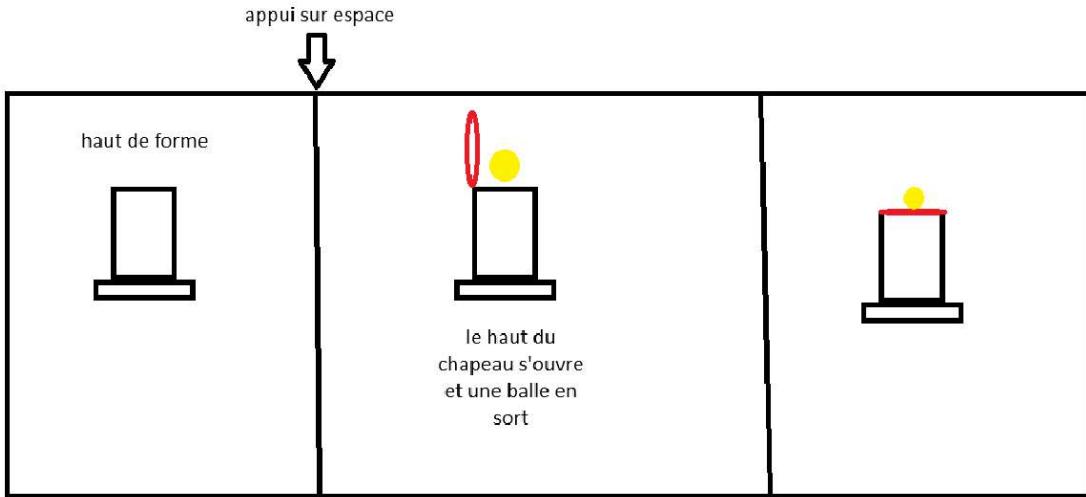
A chaque fois que la fonction est appelée, `frameAnim` augmente de 0.1. Il suffit alors de mettre cette valeur `frameAnim` (attribut de la classe `pikachu`) dans une fonction de rotation en tant qu'angle au niveau des bras, oreille et queue. Etant une rotation et non une translation, `frameAnim` ne fait qu'augmenter et ce n'est pas grave. Par exemple, 600° existe, c'est 360° (tour complet) + 240°.

4.2. Animation déclenchée au clavier

L'utilisateur peut déclencher manuellement une animation supplémentaire avec la touche espace, du clavier.

```
case ' ':
    pikachu->is_started = 1;
    printf("salut");
```

Mais tout d'abord, que fait l'animation ?



```

void Tortue::IdleAnimation() {
    frameAnim += 0.1;

    if (!is_started) return;

    if (phase == 0) {
        espanim += 2;
        poke=espanim;
        if (espanim >= 90) {
            espanim = 90;
            phase = 1;
        }
    }
    else if (phase == 1) {
        espanim -= 2;
        if (espanim>39)
            poke=espanim;
        if (espanim <= 0) {
            espanim = 0;
            phase = 0;
            is_started = 0;
        }
    }
}

```

On voit que dès que l'espace est pressé, `is_started` devient true; s'en suit :

- Le chapeau qui s'ouvre
- au même moment la balle commence à sortir,
- Une fois à la butée, le chapeau se referme
- La balle retombe sur le chapeau

La gestion des timings, étant fait au hasard et à tâtons pourrait être améliorée mais l'animation reste très sympathique. On note que Pikachu bouge encore ses bras pendant l'animation.

5. Lumières/éclairages

Dans notre projet, nous avons implémenté deux types distincts de lumières, conformément aux exigences du sujet :

1. une lumière directionnelle (`GL_LIGHT0`) ;
2. une lumière ponctuelle (`GL_LIGHT1`), attachée au haut de forme de Pikachu.

Ces deux sources agissent différemment sur le modèle et permettent d'obtenir un rendu plus vivant et mieux structuré.

Avant de configurer les différentes lumières, plusieurs états OpenGL sont activés dans le main:

```
glEnable(GL_LIGHTING);
glEnable(GL_NORMALIZE);
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
```

Explications :

- **GL_LIGHTING** active l'éclairage d'OpenGL.
- **GL_NORMALIZE** renormalise automatiquement les normales après les transformations (très important car notre Pikachu utilise beaucoup de glScalef()).
- **GL_COLOR_MATERIAL** permet à glColor3f() de définir les composantes ambiante et diffuse du matériau.
- La configuration du matériau spéculaire et de la brillance est réalisée via :

```
GLfloat matSpec[] = {1,1,1,1};
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, matSpec);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 64.0f);
```

→ Ces paramètres donnent des reflets plus nets sur certaines surfaces (joues, corps, chapeau...).

5.1 Lumière directionnelle (GL_LIGHT0)

La lumière principale du projet est définie dans la fonction gestionLumiere() :

```
glEnable(GL_LIGHT0);

GLfloat pos0[] = {1.0f, 1.0f, 0.5f, 0.0f}; // w = 0 → directionnelle
GLfloat amb0[] = {0.15f, 0.15f, 0.15f, 1.0f};
GLfloat diff0[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat spec0[] = {1.0f, 1.0f, 1.0f, 1.0f};
```

Caractéristiques :

- Le dernier paramètre $w = 0$ indique qu'il s'agit d'une lumière directionnelle, comparable au soleil.
- La lumière vient du haut et de la droite.
- Les composantes :
 - ambiante faible → éclaire légèrement toute la scène ;
 - diffuse forte → donne du volume au modèle ;
 - spéculaire forte → génère des reflets brillants.

Effet visuel :

Cette lumière assure l'éclairage global de la scène et garantit que Pikachu reste bien visible quel que soit l'angle de la caméra. Les volumes (tête, ventre, queue, chapeau) sont clairement définis.

5.2 Lumière ponctuelle (GL_LIGHT1) attachée au chapeau

La seconde lumière se trouve dans la fonction Pikachu::draw(), directement liée à la petite boule située au sommet du haut-de-forme.

C'est une **lumière ponctuelle**, car le paramètre $w = 1$:

```
glEnable(GL_LIGHT1);

GLfloat pos1[] = {
    (float)posX,
    (float)(posY + 1.8 + 0.50 + (poke * 0.015)),
    (float)posZ,
    1.0f                                     // w = 1 → Lumière ponctuelle
};

GLfloat amb1[] = {0.0f, 0.0f, 1.0f, 1.0f};
GLfloat diff1[] = {0.0f, 0.0f, 1.0f, 1.0f};
GLfloat spec1[] = {0.0f, 0.0f, 1.0f, 1.0f};
```

Spécificités :

- Lumière ponctuelle → éclaire différemment selon la distance.
- Position dynamique → elle monte et descend grâce à l'animation du chapeau (poke).
- Couleur bleue → les composantes ambiante, diffuse et spéculaire sont bleues, ce qui crée un léger halo coloré autour du haut-de-forme.

Effet visuel :

Cette source fonctionne comme un petit projecteur bleu attaché au chapeau.

Elle bouge en même temps que l'animation de la boule.

6. Architecture du projet et conclusion

En conclusion, ce projet de synthèse d'image nous a permis d'aborder l'ensemble des étapes essentielles à la création d'un personnage 3D complet, depuis la modélisation jusqu'aux animations, en passant par la gestion des textures et des lumières.

Les animations, à la fois permanentes et déclenchées par l'utilisateur, nous ont permis de comprendre vraiment en détail comment marche OpenGL. La structure modulaire de notre modèle, basée sur des blocs distincts pour chaque partie du corps, a facilité l'implémentation de ces mouvements ainsi que l'intégration des textures et des lumières.

Enfin, nous avons décidé de séparer notre projet en 3 fichiers principaux en .cpp (Cylindre, Pikachu et main) afin d'avoir une bien meilleure lisibilité sur chaque partie du programme:

- Main.cpp : gère l'affichage, une lumière, les touches du clavier, la souris, l'animation idle en grabe partie.
- Cylindre.cpp : définit une instance du cylindre en paramétrique
- Pikachu.cpp : définit bloc par bloc, les bars, le corps, la queue de pikachu, les animations et la lumière ponctuelle. C'est une classe où prend vie le pikachu.

Merci de votre lecture.