Rapport de projet : Synthèse d’images

OpenGL

Corentin PAK et Adélie FERRÉ

Lien du projet :

https://github.com/cocozer/lightcorridor

Ce rapport décrit le projet “The Light Corridor” que nous avons dû réaliser au S2 dans le cadre du cours de Synthèse d'images. Nous avons travailler en duo sur ce projet sur une durée de 2 mois environ.

Mode d’installation

Nous avons utilisé le template OpenGL d’Enguerrand pour notre projet, car Corentin ne pouvait pas faire marcher les TP openGL sur son ordinateur. Ce template nous a permis de nous mettre en place rapidement et efficacement et de créer le git.

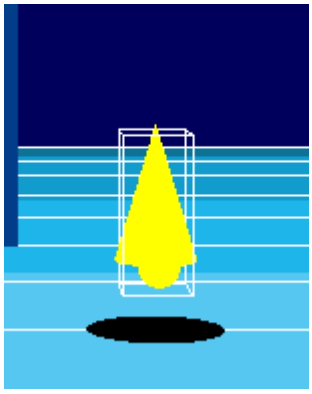
Fonctionnement

Notre projet est un jeu 3D présentant une partie ou le joueur doit avancer dans un couloir en appuyant sur la flèche du haut. Il contrôle une raquette avec la souris et doit faire rebondir une balle entre sa raquette et des obstacles qui avancent vers lui, pour donner l'impression que le joueur avance dans le couloir. Le joueur libère la balle de la raquette en faisant clique-gauche. Si la balle dépasse la raquette du joueur, alors il perd une vie. Le joueur peut rencontrer des bonus et les activer en les touchant avec la balle. Nous avons choisi ce fonctionnement pour les bonus car nous trouvions qu'il était trop facile de les activer en les touchant simplement avec la raquette. Nous avons codé 3 bonus :

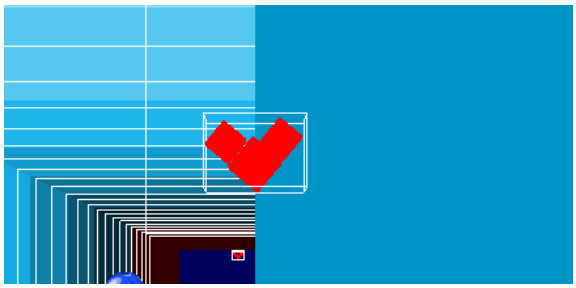
Le bonus RaquetteSticky

Ce bonus ajoute l'effet “sticky” sur la raquette : à chaque fois que le joueur touche la balle avec la raquette, celle ci

se colle et le joueur peut la relancer. Cet effet dure quelques secondes.



Le bonus Live



Ce bonus ajoute une vie en plus au joueur.

Le bonus BigRaquette

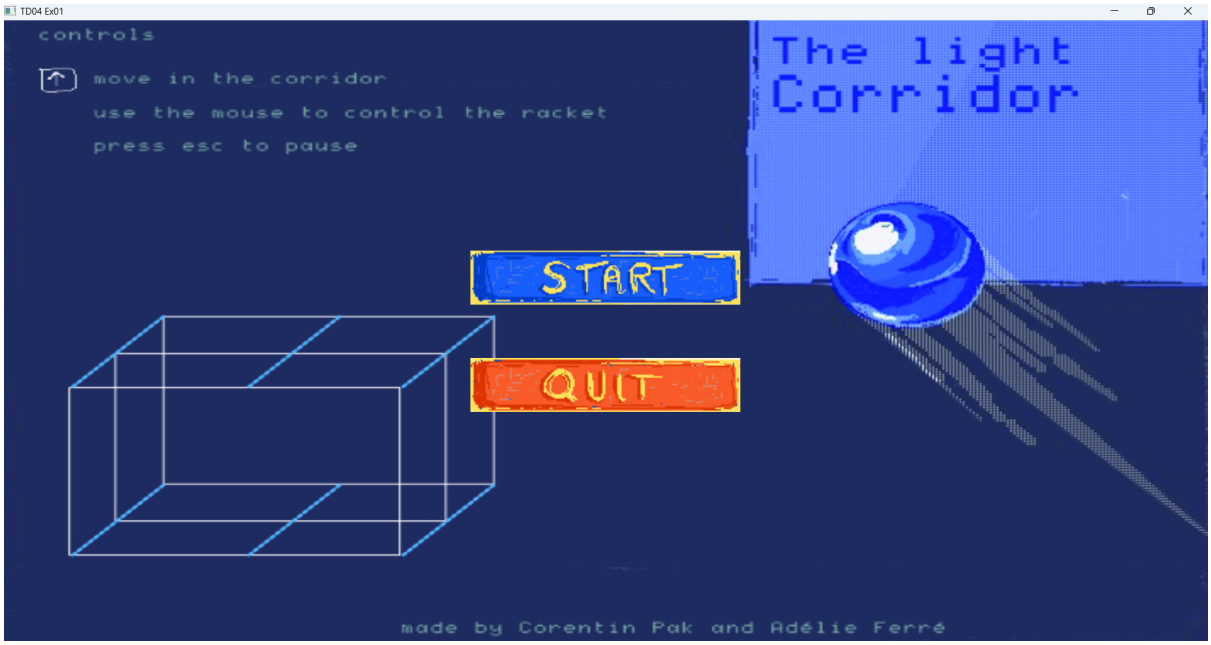
Ce “bonus” agrandit la raquette du joueur : il devient alors plus facile de frapper la balle sans perdre de vie mais le joueur est bloqué par les obstacles.

Le joueur gagne la partie lorsqu'il arrive au bout des obstacles.

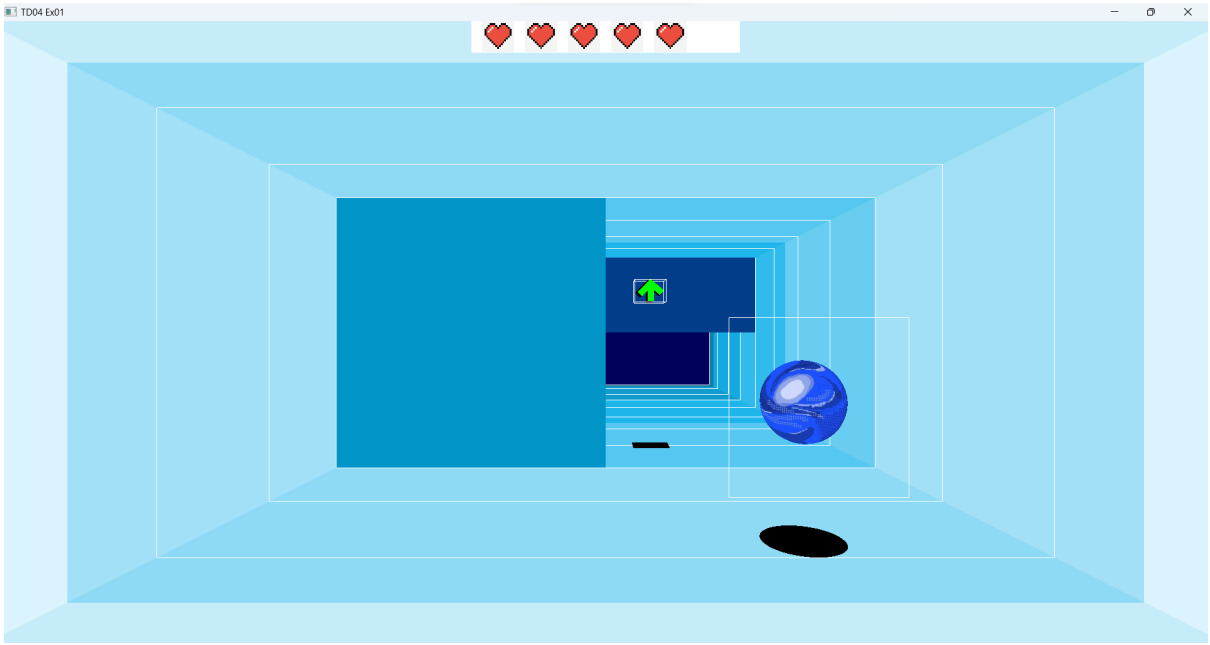
Résultats obtenus

Nous avons réussi à mettre en place une partie de ce jeu, avec quelques bonus et une cinquantaine d'obstacles à franchir. Nous avons aussi créé les menus (menu de lancement du jeu, menu pause) ou les affichages (partie gagnée/partie perdue).

Nous avons également mis en place l'affichage du nombre de vie du joueur et des bonus en cours.



Menu de lancement du jeu



Screen du gameplay

Méthode de travail

Nous nous sommes organisés grâce à un tableau de tâches sur Notion, avec des attributions et l'état de chaque tâche. Nous avons commencer le projet tôt ce qui nous a permis de ne pas manquer de temps ou rusher à la fin du projet. Nous avons

commencé le projet ensemble par la création de la balle et de la raquette, puis nous nous sommes répartis les tâches de cette façon :

Corentin : création du corridor, création du couloir, gestion du rebond de la balle sur la raquette et les murs, fonctionnement des bonus

Adélie : création des obstacles, gestion du rebond de la balle sur les obstacles, modélisation des bonus, textures

Nous avons utilisé le repository git tout au long du projet.

Détails techniques

Nous allons détailler quelques éléments de notre code.

Rebond de la balle sur la raquette

La fonction bool Ball::checkRaquetteHit(Raquette* raquette, bool raquetteSticky) permet de gérer le rebond de la ball lors de la collision avec la raquette et de renvoyer vrai ou faux si le bonus raquetteSticky est activé. Voici le pseudoCode de cette fonction :

```
Si la balle va vers le joueur
  Si la balle touche la raquette en y
    Si le x et le z de la surface de la balle touchent la raquette
      Si le bonus raquetteSticky est activé
        On retourne vrai
      FinSi
      On inverse la vitesse de la balle en y
      On calcule la direction en fonction de la distance avec le centre de la raquette
      Si le coefficient de direction dx est inférieur à 0
        le vecteur vx de la balle augmente
      Sinon
        le vecteur vx de la balle diminue
      FinSi
      Si le coefficient de direction dz est inférieur à 0
        le vecteur vz de la balle augmente
      Sinon
        le vecteur vz de la balle augmente
      FinSi
      On calcule les coefficients de direction en fonction de dx et dz
      le vecteur vx de la balle est multiplié par dx
      le vecteur vz de la balle est multiplié par dz
      Retourne Faux
    FinSi
  FinSi
FinSi
```

Effet d'optique

Pour ce qui est de la mise en place du couloir, nous avons créé plusieurs rectangles bleus orthogonaux au repères x et z, qui sont de plus en plus foncés plus ils s'éloignent de la caméra. Cela souligne visuellement la profondeur et permet de créer l'illusion qu'une source de lumière est placée au niveau de la caméra.

Un cadre blanc formé de plusieurs rectangles avance vers le joueur lorsque le bouton flèche du haut est pressé. Lorsque ce cadre dépasse le joueur, ses coordonnées reviennent à l'endroit initial, ce qui crée une boucle (voir fonction Corridor::drawCorridor).

```
212  void Corridor::drawCorridor() {
213      glColor3f(255, 255, 255); // Blanc
214      for(int i=0; i<27; i++) {
215          glPushMatrix(); // Sauvegarde de la matrice
216          float translateY = (0.2*i)-this->y;
217          while(translateY < 0) {
218              translateY +=5.4;
219          }
220              glTranslatef(0, translateY, 0);
221              drawBorderCorridor();
222          glPopMatrix(); // Reload de la matrice sauvegardée
223      }
224  }
```

fonction drawCorridor()

Nous avons choisi de faire avancer tous les éléments vers le joueur plutôt que ce soit la caméra qui avance dans les couloir : la fonction MoveCorridor qui se lance au clic du joueur sur la touche flèche-haut permet d'augmenter le y du corridor, des obstacles et des bonus pour qu'ils avancent vers nous.

```
195  void MoveCorridor(Corridor* corridor, Ball* ball, Raquette *raquette, std::vector<Obstacle>& obstacles, std::vector<Bonus>& bonuss) {
196      if(CorridorMoving && (checkRaquetteObstacleCollison(raquette, obstacles)==false)&&(!BallIsBetweenObstacleAndRaquette)) {
197          corridor->y+=0.01;
198          ball->y-=0.01;
199          for(auto & obstacle : obstacles){
200              obstacle._y-=0.01; //fait avancer l'obstacle
201          }
202          for(auto & bonus : bonuss){
203              bonus._y-=0.01; //fait avancer le bonus
204          }
205      }
206  }
```

fonction MoveCorridor()

Gestion des textures

Pour la gestion des Textures, nous avons créé la struct Texture et la fonction Texture Texture::loadTexture qui prend en paramètre une chaine de caractère vers le fichier.

```
9   struct Texture {
10      unsigned int textureID;
11      int textureWidth;
12      int textureHeight;
13      int textureChannels;
14      unsigned char* textureData;
15
16      static Texture loadTexture(const char* fileName);
17
18  };
```

struct Texture

```cpp
Texture Texture::loadTexture(const char* fileName) {
    Texture texture;
    texture.textureData= stbi_load(fileName, &texture.textureWidth, &texture.textureHeight, &texture.textureChannels, 0);
    if (texture.textureData == NULL) {
        printf("La texture n'a pas pu être chargée.\n");
    }

    glGenTextures(1, &texture.textureID);
    glBindTexture(GL_TEXTURE_2D, texture.textureID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, texture.textureWidth, texture.textureHeight, 0, GL_RGB, GL_UNSIGNED_BYTE,
    texture.textureData);
    glBindTexture(GL_TEXTURE_2D, 0);

    stbi_image_free(texture.textureData);
    return texture;
}
```

Texture Texture::loadTexture(const char* fileName)

Nous créons ainsi nos objets textures en appelant cette fonction dans le main.

```cpp
    Texture texture = Texture::loadTexture("../doc/txtureballe.jpg");
    Texture texture2 = Texture::loadTexture("../doc/fond.jpg");
    Texture textureplay = Texture::loadTexture("../doc/start.jpg");
    Texture textureexit = Texture::loadTexture("../doc/quit.jpg");
    Texture vie = Texture::loadTexture("../doc/coeur.jpg");
    Texture continue1 = Texture::loadTexture("../doc/continue.jpg");
    Texture pausefond = Texture::loadTexture("../doc/pausefond.jpg");
    Texture losescreen = Texture::loadTexture("../doc/losescreen.jpg");
    Texture winscreen = Texture::loadTexture("../doc/winscreen.jpg");
    Texture stick = Texture::loadTexture("../doc/colle.jpg");
    Texture fleche = Texture::loadTexture("../doc/fleche.jpg");
```

Nous avons également créé dans 3D_tools.cpp les fonctions permettant de mapper les textures sur les objets 3D comme drawTexturedRectangle ou encore drawTexturedSphere. Voici un exemple :

```cpp
void drawTexturedSphere(float radius, int stacks, int slices) {
    float stackStep = M_PI / stacks;
    float sliceStep = 2.0f * M_PI / slices;

    for (int i = 0; i < stacks; i++) {
        float theta1 = i * stackStep;
        float theta2 = (i + 1) * stackStep;

        glBegin(GL_QUAD_STRIP);
        for (int j = 0; j <= slices; j++) {
            float phi = j * sliceStep;

            // Vertex 1
            float x1 = radius* sin(theta1) * cos(phi);
            float y1 = radius * cos(theta1);
            float z1 = radius * sin(theta1) * sin(phi);
            glTexCoord2f((float)j / slices, (float)i / stacks);
            glVertex3f(x1, y1, z1);

            // Vertex 2
            float x2 = radius * sin(theta2) * cos(phi);
            float y2 = radius * cos(theta2);
            float z2 = radius * sin(theta2) * sin(phi);
            glTexCoord2f((float)j / slices, (float)(i + 1) / stacks);
            glVertex3f(x2, y2, z2);
        }
        glEnd();
    }
}
```

fonction drawTexturedSphere(float radius, int stacks, int slices)

Nous avons dessiner nous-mêmes les textures de boutons des menus, de la balle, de l'affichage des bonus.

Difficultés rencontrées

Nous avons rencontré quelques difficultés au cours du projet, notamment sur le gestion du rebond de la balle sur la raquette. Nous avons aussi eu au début du projet du mal à bien gérer les coordonnées de nos éléments, car nous avions fait trop de translations dans la fonction de création de l'élément. Nous avons aussi eu quelques problèmes avec git, mais cela nous a permis de mieux comprendre et utiliser l'outil par la suite.

Améliorations possibles

Avec plus de temps, nous aurions pu améliorer notre projet : par exemple en implémentant des bonus ou malus supplémentaires qui pourraient par exemple changer la taille de la balle, faire apparaître une nouvelle balle… Nous aurions également pu créer des obstacles qui bougent, ajouter plus de textures dans le jeu, ou encore créer un système de niveau ou afficher un score.

Pour conclure, ce projet a été un réel moyen de nous améliorer en c++ et en openGL. Nous avons vraiment apprécié travailler sur ce projet et sommes contents du résultat.