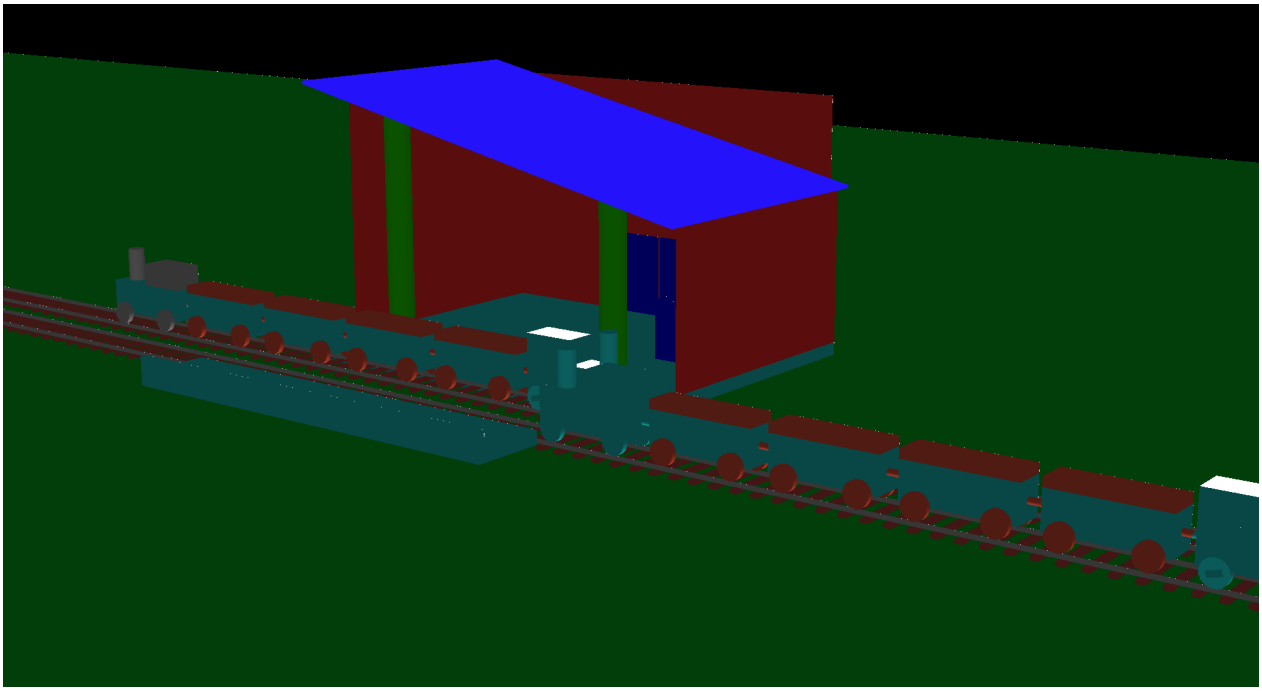
Rapport projet

GARE 3D

Fabien DUBOIS
Antoine RATO
Corentin HEMBISE

Table des matières

1 Présentation



Le projet se compile normalement grâce au Makefile. Pour avancer dans la scène, il faut activer le déplacement en appuyant sur espace, les commandes sont les suivantes :

Z : Avancer
S : Reculer
Q : Pas de côté gauche
D : Pas de côté droit
Clic droit : Monter
Clic droit : Descendre
Souris : Observer

2 Organisation

Répartition des tâches

Les tâches se sont organisées comme suit : Fabien s'est occupé de la modélisation des objets et de leur rendu. Antoine a géré la caméra en FreeFly et également fait des recherches sur les textures. Corentin s'est occupé des animations.

Organisation technique

Afin de permettre une meilleure clarté dans le code et de faciliter la collaboration, nous avons divisé notre code en différentes classes.

Pour gérer le code et ses évolutions, nous avons travaillé avec le gestionnaire de version git, les sources sont disponibles sur Github (https://github.com/dut-info/Gare-OpenGL).

3 Détails techniques

Modélisation des objets

Tous les objets étendent la classe *Objet* contenant un méthode *draw* et *modelize*. draw peut être appelé depuis l'extérieur, elle appelle la méthode modelize implémentée dans la classe fille, permettant de définir les formes de l'objet. La méthode draw nous permet d'assurer que les transformations dans modelize n'ont pas d'influence sur les autres objets, en effet, l'appel à modelize est entouré de *glPushMatrix* et *glPopMatrix*.

Les objets sont pour la plupart modélisé avec un assemblage d'objet de type *Parallelepiped, cette classe* permet de créer un parallélépipède aux dimensions souhaitées. C'est également dans cette classe que l'on calcul les normales du parallélépipède.

Gestion de la caméra

Afin de se déplacer plus facilement dans la scène, on a voulu déplacer la caméra en mode FreeFly.

Configuration

Si l'on souhaite modifier les vitesses de déplacement de la caméra ou de rotation de l'angle (souris), il faut modifier ces deux valeurs au préalable.

```
94   //Vitesse des mouvements de la caméra
95   const float g_translation_speed = 0.3; //Vitesse de la translation
96   const float g_rotation_speed = M_PI/180*0.02; //Vitesse de la rotation
```

Détails du fonctionnement de la caméra

```
56  //Declaration des fonctions
57  void Display();
58  void Reshape (int w, int h);
59  void Keyboard(unsigned char key, int x, int y);
60  void KeyboardUp(unsigned char key, int x, int y);
61  void MouseMotion(int x, int y);
62  void Mouse(int button, int state, int x, int y);
63  void Timer(int value);
64  void Idle();
65
66  //Declaration des variables et de la FreeFlyCaméra
67  Camera g_camera; //Objet de Camera.cpp
68  bool g_key[256]; //Touches disponibles
69  bool g_shift_down = false; //Booleen pour vérifier si la touche est appuyée(true) ou non(false)
70  bool g_fps_mode = false; //Boolean pour vérifier si le FPS (Camera FreeFly) est activée ou non.
71  int g_viewport_width = 0; //Largeur de la fenetre
72  int g_viewport_height = 0; //Hauteur de la fenetre
73  bool g_mouse_left_down = false; //Booleen pour vérifier les mouvements de la souris
74  bool g_mouse_right_down = false; //Booleen pour vérifier les mouvements de la souris
```

La première partie est la déclaration des fonctions, pour pouvoir les appeler dans un ordre quelconque. La seconde partie est la déclaration des variables utilisées pour la caméra FreeFLy (la caméra y compris).

```
205 void Keyboard(unsigned char key, int x, int y)
206 {
207     if(key == 27) { //Si la touche appuyé est la touche 27 (Echap), alors quitter le programme
208         exit(0);
209     }
210
211     if(key == ' ') { //Si la touche appuyé est la touche espace, inveré le mode Freely (sortir et entrer en mode FPS)
212         g_fps_mode = !g_fps_mode;
213
214         if(g_fps_mode) {
215             glutSetCursor(GLUT_CURSOR_NONE); //Cacher le curseur
216             glutWarpPointer(g_viewport_width/2, g_viewport_height/2); //Appliquer le scale
217         }
218         else {
219             glutSetCursor(GLUT_CURSOR_LEFT_ARROW); //Afficher le curseur
220         }
221     }
222
223     //Continuer l'action si la touche est toujours appuyé
224     if(glutGetModifiers() == GLUT_ACTIVE_SHIFT) {
225         g_shift_down = true;
226     }
227     else {
228         g_shift_down = false;
229     }
230
231     g_key[key] = true;
232 }
```

Configuration du clavier lors de l'appuie d'une touche et du relâchement (keyboardUp).

Configuration de l'action des touches lorsqu'elles sont actionnées.

```
271  void Mouse(int button, int state, int x, int y)
272  {
273      if(state == GLUT_DOWN) {                    //Si l'un des clics de la souris est appuyé, alors :
274          if(button == GLUT_LEFT_BUTTON) {        //Si c'est le clic gauche,
275              g_mouse_left_down = true;           //Mettre le booleen clic gauche à true
276          }
277          else if(button == GLUT_RIGHT_BUTTON) {  //Sinon si c'est le clic droit,
278              g_mouse_right_down = true;          //MEttre le booleen clic droit à true
279          }
280      }
281      else if(state == GLUT_UP) {                 //Si l'un des clics de la souris est relaché, alors :
282          if(button == GLUT_LEFT_BUTTON) {        //Si c'est le clic gauche,
283              g_mouse_left_down = false;          //Mettre le booleen clic gauche à false
284          }
285          else if(button == GLUT_RIGHT_BUTTON) {  //Sinon si c'est le clic droit,
286              g_mouse_right_down = false;         //Mettre le booleen de clic droit à false
287          }
288      }
289  }
```

Adapté de: http://nghiaho.com/?p=1613

Gestion des animations

Solution initiale

Nous avons tenté de modéliser les rails en fonction d'une courbe de Bézier et d'appliquer des transformations sur le train pour le faire suivre les rails.

L'idée est de créer une classe BezierCurve prenant à la construction une liste de points et permettant à partir d'une valeur de « t » d'obtenir le point correspondant (fig. 2). Ainsi, il aurait suffit d'appliquer une translation au train suivant les valeurs du point. Pour que le mouvement soit réaliste, il faut que le train suive une rotation dépendante de la tangente de la courbe (fig. 1). Pour cela, il faut une seconde fonction qui calcule la tangente en un point précis.

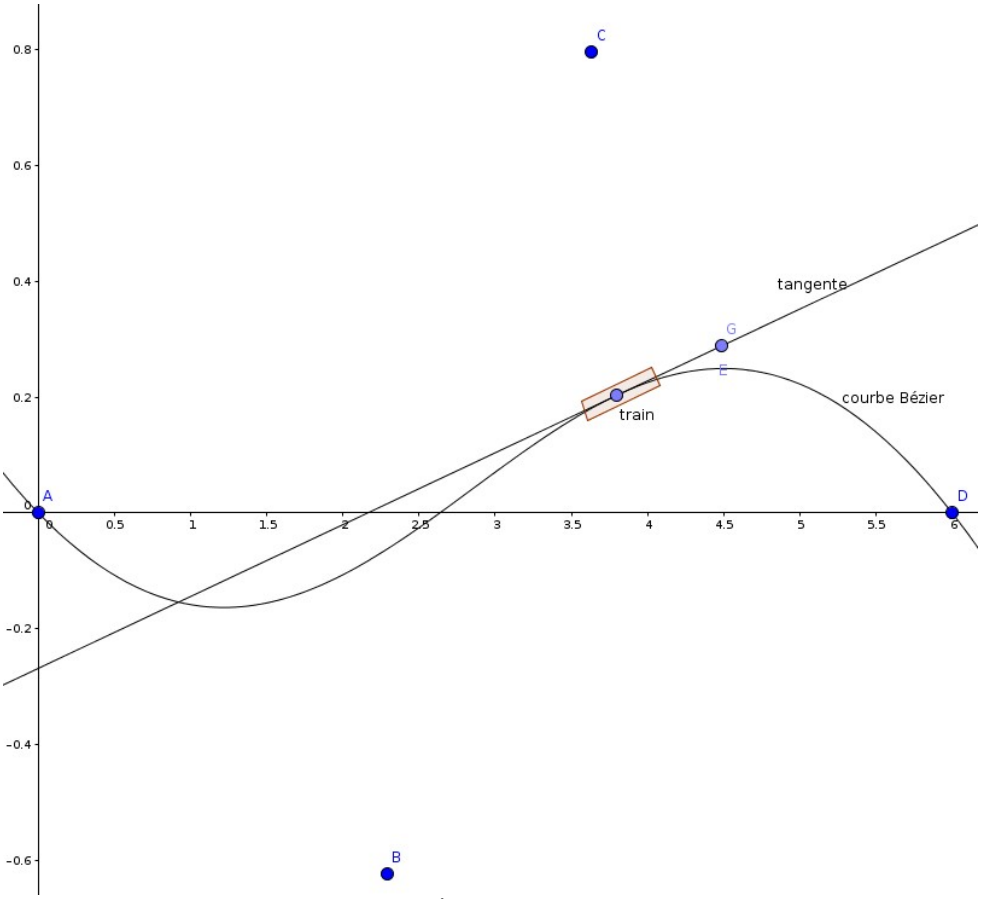


Illustration 1: Courbe de Bézier

Quelques traces de nos tentatives sont dans la classe BezierCurve, mais nous n'avons pas pu finaliser le travail.

```
Point BezierCurve::getPoint(double t)
{
  std::vector<Point> tmp = std::vector<Point>(this->points);

  for (int i = tmp.size() - 1; i > 0; i--) {
    for (int k = 0; k < i; k++)
        tmp[k] = tmp[k] + ( tmp[k+1] - tmp[k] ) * t;
  }

  Point result = tmp[0];
  //delete tmp;
  return result;
}
```

Illustration 2: Méthode de calcul d'un point sur une courbe de Bézier

Solution alternative

Finalement, nous avons géré l'animation du train seulement en ligne droite, nous translatons le train sur un axe du repère. Pour que le train ne s'arrête pas trop brusquement en gare, à l'arrivée devant la gare, nous avons fais dépendre la position du train, de la fonction :

$\frac{1+\cos(\frac{1}{4}x\times\pi)}{2}$ représentation (fig. 3)

Ceci nous permet d'obtenir une décélération plus fluide. Les roues sont également en rotation via ce même paramètre.

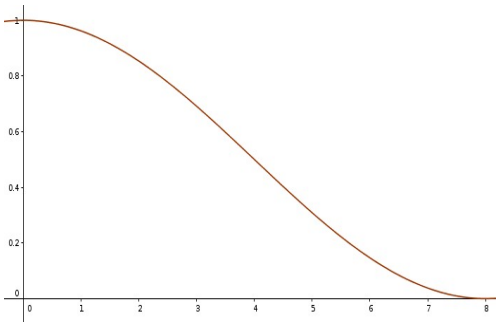


Illustration 3