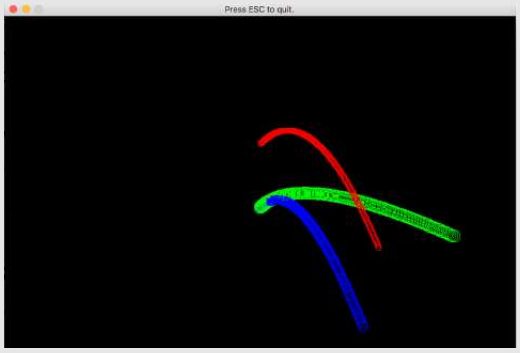
IN 103 - TD 03

Solution

Dans ce TD, on souhaite développer un petit moteur de physique simple en 2 dimensions. Le moteur doit être capable de simuler le mouvement de corps donnés sur lesquels s'appliquent des forces également données.

L'affichage donnera lieu à une animation des mouvements. L'image ci-dessous montre un résultat attendu, en ayant laissé la trace des positions successives pour le besoin de l'illustration statique.



1 Les corps

Q1 Sachant qu'un corps est caractérisé par :

- sa couleur de dessin,
- le rayon du cercle qui le représente à l'écran,
- sa masse,
- ses coordonnées x et y,
- sa vitesse en x et en y,
- son accélération en x et en y,

déduisez-en une structure de données pour le représenter.

Solution

À noter que le rayon est un `int` car il n'y a pas de fraction de pixel sur un écran. Pour ce qui est de la couleur, elle est informatiquement représentée par la juxtaposition de 3 valeurs entre 0 et 255 inclus (donc des octets) représentant la proportion de rouge, de vert et de bleu dans ladite couleur.

```
/** Structure of a body. */
struct body_t {
  int color ;       /** Drawing color. */
  int radius ;      /** Radius of the circle used to draw the body. */
  double mass ;     /** Mass in Kg. */
  double x, y ;     /** 2D coordinates. */
  double vx, vy ;   /** 2D speed. */
```

```
  double ax, ay ;  /** 2D acceleration. */
} ;
```

Q2 Sachant que l'on souhaite pouvoir rajouter ou supprimer dynamiquement des corps dans la simulation, proposez une structure de données pour mémoriser l'ensemble des corps d'une simulation.

Solution

Une liste chaînée de structures `struct body_t` sera très pratique.

```
/** Linked list of bodies. */
struct body_list_t {
  struct body_t *body ;
  struct body_list_t *next ;
} ;
```

2 Les forces

Q3 On souhaite permettre l'appliction de forces non constantes, possiblement dépendantes de la position, de la vitesse du corps sur lequel elles s'appliquent. Proposez une représentation d'une force.

Solution

Si une force peut ne pas être une constante . . . c'est donc une fonction qui prend en argument la position et la vitesse courante du corps sur lequel elle s'exerce. Ces arguments sont bien entendu des flottants puisqu'ils représentent des grandeur physiques continues.

Q4 Vu qu'une force est caractérisée en 2 dimensions par ses composantes x et y, comment pouvez-vous retourner ces deux composantes aux fonctions qui ont besoin de travailler avec une force ?

Solution

Il suffira qu'une fonction représentant une force ≪retourne≫ ces deux valeurs par adresse. Ainsi, une ≪fonction de force≫ aura pour prototype :

void f (**double** x, **double** y, **double** vx, **double** vy, **double** ∗fx, **double** ∗fy);

Bien entendu, la réponse de **Q4** nous indique que nous définirons nos forces statiquement. Nous allons les stocker un tableau statique terminé par `NULL` (qui nous servira pour savoir quand nous aurons atteint la fin lors des parcours.

Dans le fichiers `body.h`, `draw.h` et `simu.h` vous sont données les structures de données que les questions précédentes vous invitaient à concevoir.

En outre se trouvent les prototypes des différentes définitions que vous devrez écrire. Utilisez donc ces fichiers d'en-tête pour la suite.

3 Structure de l'algorithme de simulation

Le moteur de simulation va donc appliquer les lois de la physique newtonienne en boucle, chaque itération de la boucle faisant avancer le ≪temps≫ (et affichant la position des corps). Il convient donc de définir un ≪delta de temps≫ qui représentera le temps physique écoulé entre deux étapes successives de la simulation. Dans la suite nous le nommerons `DT`. Sa valeur sera à votre discrétion.

Q5 Puisque notre objectif est de simuler le mouvement de corps soumis à des forces, quelle va être la forme de l'algorithme ?

Solution

Nous avons donc en entrée des corps et des forces. Nous voulons voir l'évolution de la position d'un corps au cours du temps. La position évolue avec la vitesse. La vitesse évolue avec l'accélération. L'accélération dépend des forces appliquées au corps. Accessoirement, il faudra effacer l'écran au début et dessiner les corps à leur nouvelle position à la fin. Donc il va falloir :

1. effacer l'écran,
2. calculer pour chaque corps l'accélération qu'il subit,
3. calculer pour chaque corps sa vitesse,
4. calculer pour chaque corps sa position,
5. dessiner chaque corps.

4 Manipulation des corps

Q6 Écrivez la fonction `new_body` qui crée un nouveau corps et l'ajoute à la structure de mémorisation déduite en question **Q2**.

Solution

Cette fonction doit prendre en paramètre les différentes caractéristiques qui représentent un corps ainsi que la liste dans laquelle chaîner le corps.

C'est donc une simple fonction d'ajout en tête à une liste chaînée. Il faut donc allouer la mémoire pour la structure de corps, puis allouer la mémoire pour la chaînon de liste, initialiser le corps, l'insérer dans le chaînon et chaîner ce dernier.

Le code est lisible dans le listing de réponse à la question **Q8**.

Q7 Quelles sont les équations qui permettent de calculer le nouvelle vitesse et la nouvelle position d'un corps ?

Solution

Pour la vitesse, il suffit d'intégrer l'accélération par rapport au temps. Pour la position, il suffit d'intégrer la vitesse par rapport au temps.

$$\begin{cases} \mathcal{V}_x(t) &= \mathcal{V}_x(t-1) + \mathcal{A}_x(t) \times \texttt{DT} \\ \mathcal{V}_y(t) &= \mathcal{V}_y(t-1) + \mathcal{A}_y(t) \times \texttt{DT} \\ \\ \mathcal{X}(t) &= \mathcal{X}(t-1) + \mathcal{V}_x(t) \times \texttt{DT} \\ \mathcal{Y}(t) &= \mathcal{Y}(t-1) + \mathcal{V}_y(t) \times \texttt{DT} \end{cases}$$

Q8 Écrivez la fonction `move_body` qui met à jour la vitesse et la position du corps qu'elle reçoit en argument.

Solution

```
   -------------- body.c --------------
```

```c
#include <stdlib.h>
#include "body.h"

/** Integration delta time in seconds. */
#define DT (0.05)



/** Create and register in a linked list a new body. [l] is the linked list
    of bodies where to add this body. The other parameters are the various
    properties of teh body.
    This function return the new head of bodies linked list. */
struct body_list_t* new_body (struct body_list_t* l, int color, int radius,
                              double mass,
                              double x, double y, double vx, double vy)
{
  struct body_list_t *new_cell ;
  struct body_t *b = malloc (sizeof (struct body_t)) ;
  if (b == NULL) return (NULL) ;

  /* Allocate a new linked list cell. */
  new_cell= malloc (sizeof (struct body_list_t)) ;
  if (new_cell == NULL) {
    free (b) ;   /* Free the successfully allocated body. */
    return (NULL) ;
  }

  /* Initialize the body. */
  b->color = color ;
  b->radius = radius ;
  b->mass = mass ;
  b->x = x ; b->y = y ;
  b->vx = vx ; b->vy = vy ;
  b->ax = 0.0 ; b->ay = 0.0 ;

  /* Link the body in the list. */
  new_cell->body = b ;
  new_cell->next = l ;

  return (new_cell) ;
}



/** Move a body according to its current acceleration and speed.
    This function updates the [vx], [vy], [x], [y] components. It simply
    integrate acceleration and speed. */
void move_body (struct body_t *b)
{
  /* Update speed from acceleration. */
  b->vx = b->vx + b->ax * DT ;
  b->vy = b->vy + b->ay * DT ;
  /* Update position from speed. */
  b->x = b->x + b->vx * DT ;
  b->y = b->y + b->vy * DT ;
}
```

Désormais, nous savons mettre à jour la vitesse et la position d'un corps à partir de l'accélération qu'il subit. Il nous faut donc calculer cette accélération à partir des forces appliquées.

5 Manipulation des forces et simulation

Q9 Définissez autant de forces que vous souhaitez et stockez-les dans un tableau.

Solution

Comme je suis relativement fainéant, je n'en définis qu'une dans la solution. Notez que je n'ai pas mis la taille du tableau dans sa définition : je laisse ce travail au compilateur qui voit bien avec combien d'éléments j'ai initialisé le tableau. Il en déduit sa taille.

```
/** A simple constant force. It does not depend on the position or speed.
    The coordinates of the computed force are returned by address in
    [fx] and [fy]. */
void f (double x, double y, double vx, double vy, double *fx, double *fy)
{
  *fx = 5.0 ;
  *fy = 5.0 ;
}

/** The array containing all the forces to apply to bodies. It must be
    NULL-terminated.
    We could imagine that all the forces do not apply on all the objects.
    In this case, make several arrays or link objects and forces. */
force_t forces_array[] = { f, NULL } ;
```

Q10 Quelles sont les équations qui régissent l'accélération subie par un corps en fonction de la force qu'il subit ?

Solution

C'est simplement la seconde loi de Newton. Vu qu'il y a plusieurs forces, il faut accumuler les accélérations.

$$\begin{cases} \mathcal{A}_x(t) &= \mathcal{A}_x(t-1) + \mathcal{F}_x(t)/m \\ \mathcal{A}_x(t) &= \mathcal{A}_y(t-1) + \mathcal{F}_y(t)/m \end{cases}$$

Q11 Écrivez la fonction `apply_forces_on_body` qui prend en argument un corps, les forces et applique chacune des forces sur ce corps pour mettre à jour l'accélération de ce corps.

Solution

Le code est lisible dans le listing de réponse à la question **Q13**.

Q12 Et la gravité ? Comment pouvons nous intégrer la gravité dans notre moteur ?

Solution

La première solution est de calculer la force induite entre la Terre et le corps du fait de leur distance et leur masse. Mais ce n'est pas le plus simple car il faudrait déjà rajouter autant de forces dans notre tableau qu'il y a d'objets. Et en plus, on trouverait toujours une accélération de -9.81 m/s^2.

Autant directement partir du principe que chaque corps a initialement cette accélération.

Q13 Maintenant que nous savons intégrer la gravité dans la simulation, écrivez une fonction `simulate_bodies` qui prend en arguments les corps et les forces et applique les forces et met à jour la position de chaque corps.

Solution

```
   -------------- simu.c --------------

#include <stdlib.h>

#include "body.h"
#include "simu.h"


/** A simple constant force. It does not depend on the position or speed.
    The coordinates of the computed force are returned by address in
    [fx] and [fy]. */
void f (double x, double y, double vx, double vy, double *fx, double *fy)
{
  *fx = 5.0 ;
  *fy = 5.0 ;
}



/** The array containing all the forces to apply to bodies. It must be
    NULL-terminated.
    We could imagine that all the forces do not apply on all the objects.
    In this case, make several arrays or link objects and forces. */
force_t forces_array[] = { f, NULL } ;




/** Apply all the forces of tha array [forces] onto the body [b]. This function
    simply update the acceleration from the applied force, the current
    acceleration of the body and its mass. It's indeed the second Newton's
    law. */
void apply_forces_on_body (struct body_t *b, force_t *forces)
{
  unsigned int i = 0 ;

  /* Scan the array of forces up to its end. */
  while (forces[i] != NULL) {
    double fx, fy ;

    /* Apply the force by calling the force-function, depending on the body's
       position and speed. */
    (forces[i]) (b->x, b->y, b->vx, b->vy, &fx, &fy) ;
    /* Update the body's acceleration to reflect the force's effect. */
    b->ax = b->ax + fx / b->mass ;
    b->ay= b->ay + fy / b->mass ;
    /* Going to the next force if any. */
    i++ ;
  }
}



/** Run all the forces on all the bodies. Add the gravity as a shortcut
    by setting the initial y acceleration to 1g. This is simpler than computing
    the force induced between the Earth and each object, then apply this force
    to compute the acceleration. */
void simulate_bodies (struct body_list_t *bl, force_t *forces)
{
  while (bl != NULL) {
    struct body_t *b = bl->body ;
```

```
      /* Clear the x acceleration and initialize y with the gravity. */
      b->ax = 0.0 ;
      b->ay = -9.81 ;
      /* Apply all the forces on this body. */
      apply_forces_on_body (b, forces) ;
      /* Update speed and position. */
      move_body (b) ;

      bl = bl->next ;
    }
  }
```

Les fichiers `draw.(c|h)` et `physics` vous sont donnés. Le premier contient une fonction `draw_bodies` permettant d'afficher (ou d'effacer) tous les corps, le second contient le `main`. Le `main` s'attend à disposer de la fonction `simulate_bodies` que vous avez écrite à la question précédente.

Q14 Compilez votre programme et testez-le. Pour compiler, vous devrez utiliser la ligne de commande ci-dessous, en rajoutant les fichiers `.c` que vous aurez écrits à ceux qui vous sont fournis.

```
gcc gfxprims.c VOS_FICHIERS draw.c physics.c `sdl-config --cflags --libs`
```