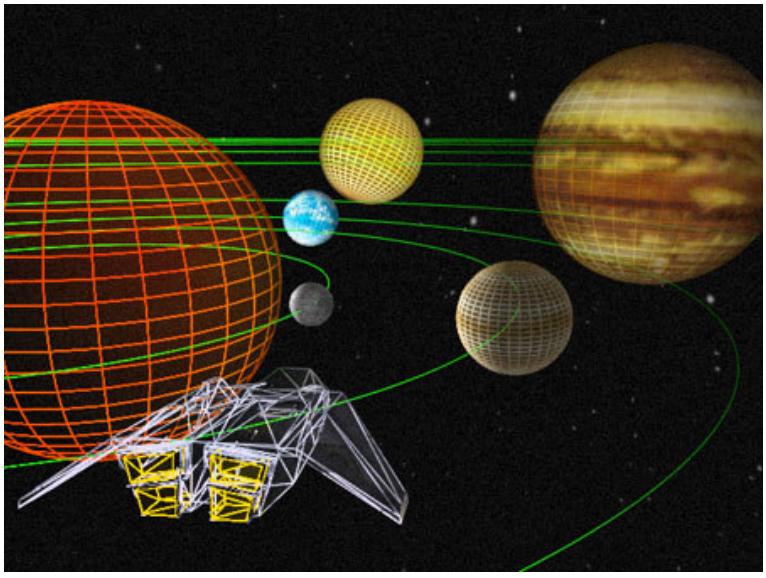
Informatique géométrique et graphique

Visualisation du système solaire
SolarSys v1.0 (OpenGL / Glut)

FUCHS Sébastien
VOURIOT Thierry
Maîtrise d'Informatique
19 mai 2003
Groupe 1



SolarSys v1.0 screenshot

Sommaire :

Introduction page 3

- Sujet page 3
- Présentation du système solaire page 4

Développement page 9

- Langage de programmation et structure globale page 9
- Modélisation du système solaire page 12
- Caméras page 12
- Calcul des trajectoires des planètes page 13
- Vaisseau spatial page 14
- Sélection des planètes page 15

Code source page 16

- CObject page 16
- CCamera page 17
- CPlanet page 17
- CSpaceShip page 19
- CSkyBox page 20
- CSolarSystem page 21
- CTga page 21

Manuel utilisateur page 23

Photos d'écran page 24

Conclusion page 25

Informations personnelles page 26

Introduction

Sujet :

Il s'agit de faire une visualisation du système solaire (soleil + planètes + satellites) grâce à un programme OpenGL. Les planètes seront assimilées à des sphères qui devront suivre des trajectoires les plus réalistes possibles. Les planètes du système solaire devront se mouvoir, la caméra devra pouvoir se déplacer de façon interactive. Différentes options pourront être données au programme : (au moins deux devront être faites)

1. Possibilité de voir les trajectoires déjà parcourues par les astres sous forme d'une courbe.
2. Calcul des trajectoires des planètes avec la loi de gravitation universelle, on pourra simplement utiliser le schéma d'Euler. Quelle est la principale différence par rapport à des trajectoires circulaires ?
3. Commande interactive d'un vaisseau spatial.
4. Visualisation du système solaire depuis la Terre. Cette visualisation devra permettre de visualiser la position des astres dans le ciel.

Le rapport devra absolument contenir :

- Un manuel permettant d'utiliser le programme
- Une explication du fonctionnement du programme. En particulier on détaillera les transformations géométriques entrant en jeu.

Indications :

Pour avoir une scène animée il faut utiliser la fonction `glutIdle`
Quelques données sur le système solaire :

- Rayon Soleil : 696,0*10^6m, Terre : 6,38*10^6m, Lune : 1,74*10^6m, Vénus : 6,05*10^6m
- Distance Soleil-Terre : 149,6*10^9m, Soleil-Vénus : 108,2*10^9m, Terre-Lune : 384,4*10^6m
- Période de rotation : Terre 365 jours, Lune 29 jours, Vénus 225 jours.
- G = 6,67e – 11
- Période de rotation de la Terre sur elle-même : 86200 s, angle de l'axe de rotation de la Terre 23.4°

Ces dimensions ne seront pas forcément à la même échelle, on pourra par exemple multiplier par 1000 le rayon des planètes et par 50 le rayon du soleil et les distances entre Terre et Lune, mais ces échelle devront être connues de l'utilisateur.

Présentation du système solaire :

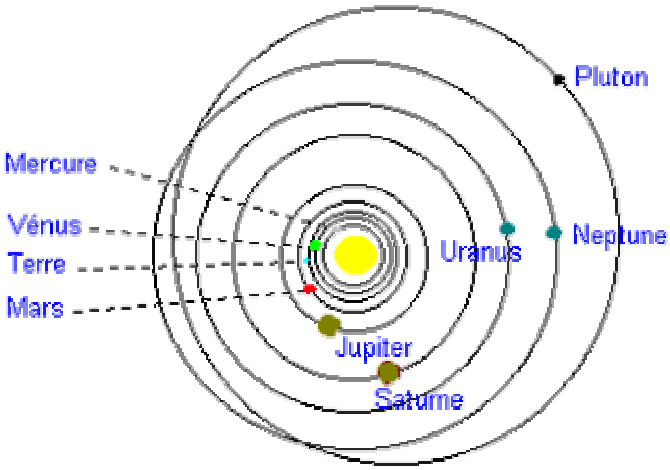
LE SYSTEME SOLAIRE

Le système solaire est constitué d'un ensemble de planètes qui tournent autour du Soleil, dans le même sens, sensiblement sur le même plan, mais à des distances très diverses.

Le mot planète signifie 'errant'. Les planètes sont effectivement des objets celestes errants dans un ciel d'étoiles fixes.

Les planètes du système solaire sont connues pour la plupart depuis la plus haute antiquité et sont citées dans l'ordre suivant :La Terre, La Lune, Mercure, Vénus, Mars, Jupiter, Saturne, Neptune, Uranus et Pluton.

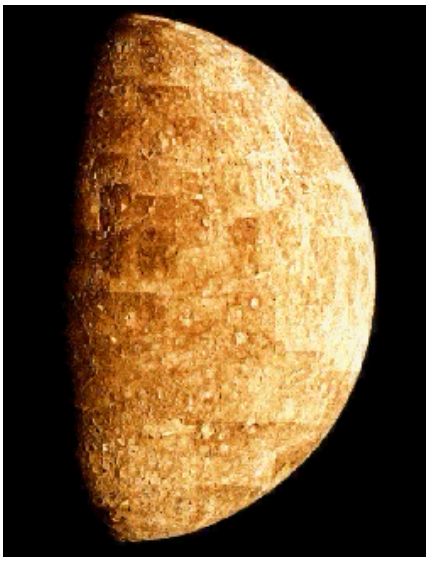
Le dessin ci-dessous donne une vision schématique (et fausse!) du système solaire.



TERRE

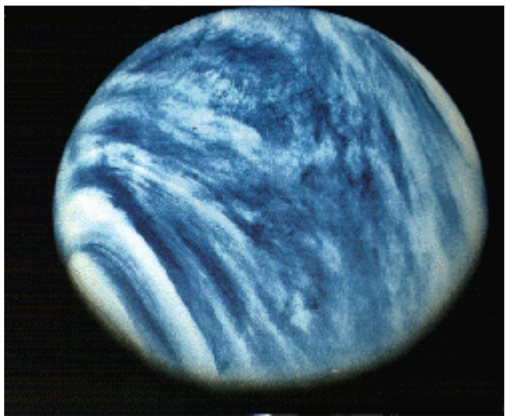
Diamètre polaire : 12756 km
Diamètre équatorial : 12713 km
Période de rotation sidérale:23h 56mn 4s
Inclinaison de l'équateur sur l'écliptique : 23°26'
Masse : 5.98 10^{24}kg
Demi-grand axe de l'orbite : 149 897 570 km
Excentricité : 0.0176
Période de révolution sidérale : 365j 6h 9mn 9.5s

MERCURE



Diamètre équatorial : 4878km
Masse / Terre : 0.055
Période de rotation sidérale : 58j 15h 38mn
Demi-grand axe de l'orbite : 0.387 UA
Excentricité : 0.206
Inclinaison sur l'écliptique : 7°00'
Période de révolution sidérale : 87.969j
Période de révolution synodique : 115.9j

VENUS



Diamètre équatorial : 12 104km
Masse par rapport à la Terre : 0.815
Période de rotation sidérale : 243.01j
Demi-grand axe de l'orbite : 0.7233 UA
Excentricité : 0.007
Inclinaison de l'orbite sur l'écliptique : 3°23'
Période de révolution sidérale : 224.701j
Période de révolution synodique 583.92j

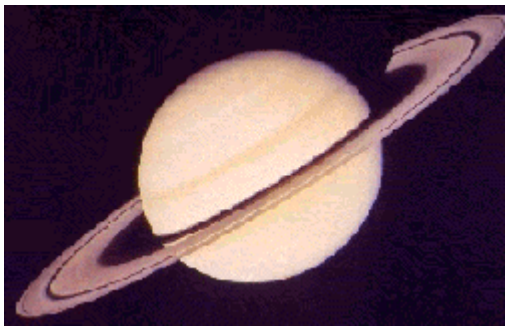
MARS

Diamètre équatorial : 6794km
Diamètre polaire : 6760km
Masse / Terre : 0.107
Période de rotation sidérale : 24h 37mn 22.7s
Demi-grand axe de l'orbite : 1.5337 UA
Excentricité 0.093
Inclinaison sur l'écliptique : 1°51'
Période de révolution sidérale : 686.980j
Période de révolution synodique : 779.94j

JUPITER

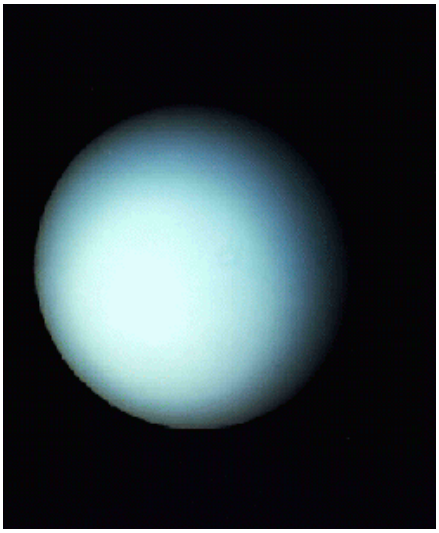
Jupiter est entouré de 16 satellites, les quatre plus importants étant appelés satellites galiléens. Ce sont Europe (au centre), Io (en haut à gauche), Callisto (en bas à gauche) et Ganymède (en bas à droite).
Diamètre équatorial : 142 796 km (11,9 fois celui de la Terre)
Diamètre polaire : 133 540 km
Masse / Terre : 317.95
Période de rotation sidérale : 9h 50 mn
Demi-grand axe :5.2026UA
Excentricité : 0.048
Inclinaison sur l'écliptique : 1°18'28"
Période de révolution sidérale : 11 ans 314.84 j
Période de révolution synodique : 1 an 3 j

SATURNE



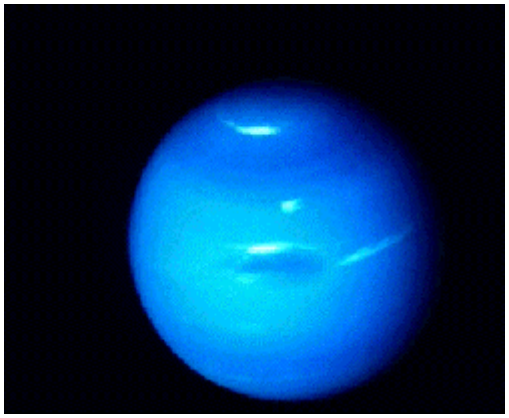
Diamètre équatorial : 120 000 km (9.4 fois celui de la Terre)
Diamètre polaire : 108 000 km
Masse / Terre : 95.2
Période de rotation sidérale : 10 h 14 mn
Demi-grand axe de l'orbite : 9.5547 UA
Excentricité : 0.056
Inclinaison sur l'écliptique : 2°30'
Période de révolution sidérale : 29 ans 167 j
Période de révolution synodique : 1 an 13 j

URANUS



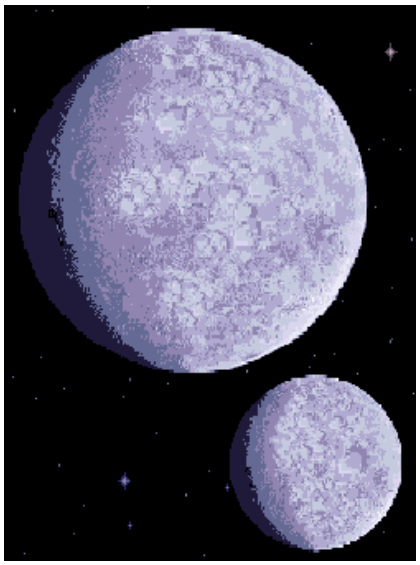
Diamètre équatorial : 51 200 km (4 fois celui de la Terre)
Masse / Terre : 14.6
Demi-grand axe : 19.218 UA
Excentricité : 0.047
Inclinaison sur l'écliptique 0°46'
Période de révolution sidérale : 84 ans 7.4 j
Période de révolution synodique 1 an 4.6 j

NEPTUNE



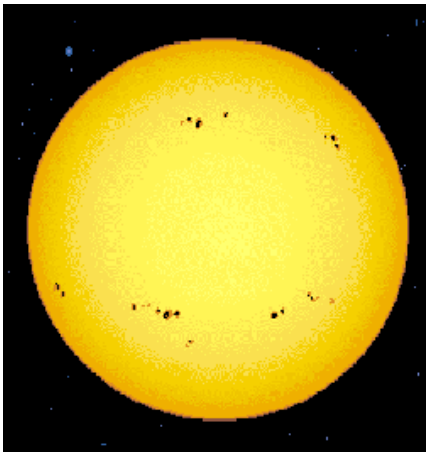
Diamètre équatorial : 48 600 km (3.81 fois celui de la Terre)
Masse / Terre : 17.2
Période de rotation sidérale : 18.2 h
Demi-grand axe : 30.1096 UA
Excentricité : 0.009
Inclinaison sur l'écliptique : 1°47'
Période de révolution sidérale : 164 ans 280.3j
Période de révolution synodique : 1 an 2.2 j

PLUTON



Pluton et Charon
Diamètre équatorial : 2200 km (0.17 fois celui de la Terre)
Masse / Terre : 0.002
Période de rotation sidérale : 6j 9h 18 mn
Demi-grand axe : 39.4387 UA
Excentricité : 0.246
Inclinaison sur l'écliptique : 17°10'
Période de révolution sidérale :247 ans 249 j
Période de révolution synodique : 366,74 j

SOLEIL



Diamètre équatorial : 696 000 km
Masse / Terre : 333 000
Période de rotation sidérale : 25.38 j h

Données relatives aux planètes du système solaire :

Nom	Demi-grand axe de l'orbite (UA)	Distance moyenne au Soleil (10^6 km)	Excentricité de l'orbite	Inclinaison de l'orbite sur l'écliptique	Durée de la révolution sidérale	Diamètre équatorial (Terre = 1)	Diamètre équatorial (km)
Mercure	0,387	57,9	0,206	7° 00'	88 j	0,38	4 878
Vénus	0,723	108,2	0,007	3° 23'	225 j	0,95	12 104
Terre	1,000	149,6	0,016	0° 00'	1,00 an	1,00	12 756
Mars	1,524	229,0	0,093	1° 51'	1,88 an	0,53	6 795
Jupiter	5,210	779,4	0,050	1° 18'	11,87 an	11,21	142 985
Saturne	9,585	1433,9	0,056	2° 29'	29,48 an	9,45	120 537
Uranus	19,233	2877,2	0,044	0° 46'	84,07 an	4,01	51 119
Neptune	30,110	4504,4	0,011	1° 46'	164,90 an	3,96	50 538
Pluton	39,254	5872,3	0,244	17° 09'	247,85 an	0,18	2320
Soleil						109,12	1 392 000

Nom	Vitesse orbitale moyenne ($km.s^{-1}$)	Vitesse de libération ($km.s^{-1}$)	Masse (Terre = 1)	Densité (eau = 1)	Gravité en surface (Terre = 1)	Durée de la rotation sidérale	Inclinaison de l'équateur sur l'orbite	Albédo
Mercure	47,9	4,3	0,055	5,4	0,39	58 j 16 h	0° 00'	0,11
Vénus	35,0	10,4	0,815	5,2	0,90	243 j ***R***	177° 18'	0,65
Terre	29,8	11,2	1,000	5,5	1,00	23 h 56 min	23° 17'	0,37
Mars	24,1	5,0	0,107	3,9	0,38	24 h 37 min	25° 11'	0,15
Jupiter	13,1	60,2	317,9	1,3	2,64	9 h 55 min	3° 07'	0,52
Saturne	9,6	32,3	95,2	0,7	1,16	10 h 39 min	26° 43'	0,47
Uranus	6,8	22,5	14,6	1,3	1,17	15 h 35 min	97° 51'	0,51
Neptune	5,4	23,9	17,2	1,8	1,20	18 h 25 min	29° 33'	0,41
Pluton	4,7	1,2	0,003	1,1	2,03	153 h 16 min	118° 00'	0,30
Soleil		619	332 946	1,4	28	25 à 35 j [(1)]		

[(1)] 25 jours à l'équateur et 35 jours aux pôles ***R*** : sens rétrograde

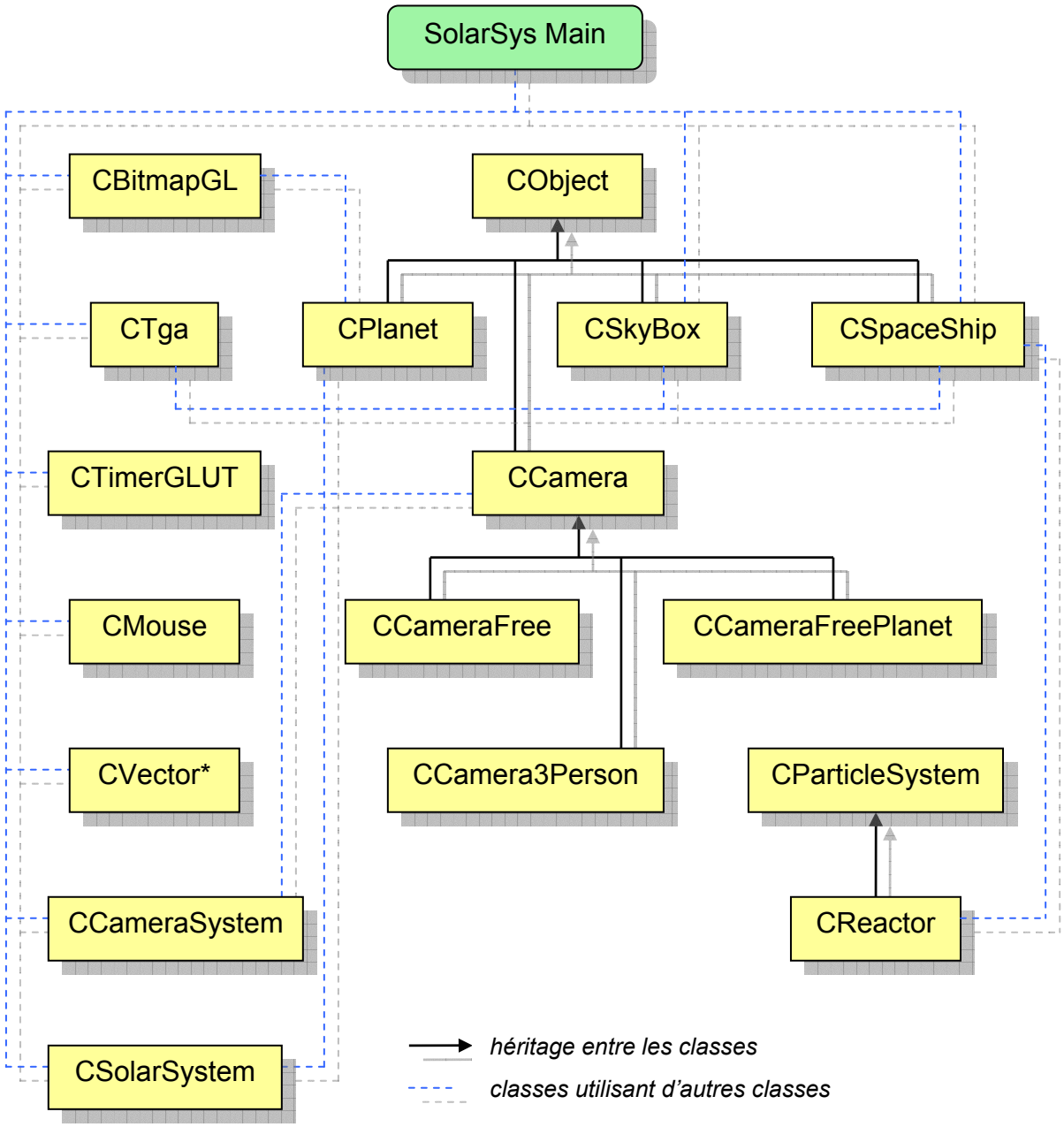
Développement

Langage de programmation et structure globale :

Le langage de programmation utilisé pour développer SolarSys est le C++. L'orientation objet de ce langage permet une meilleure représentation des composants du système solaire ; planètes, satellites, vaisseau, etc…

La représentation en trois dimensions utilise OpenGL couplé au toolkit GLUT. GLUT est ensemble d'outils destinés à faciliter le développement d'applications OpenGL, il permet notamment la gestion des fenêtres, du clavier et de la souris. Bien que d'une puissance limitée GLUT à l'avantage d'être extrêmement facile à programmer, de plus il permet une portabilité à 100% du code (quel que soit le système d'exploitation, tous les programmes seront parfaitement compilables sans aucune modification à apporter).

Un ensemble de classes a été écrite pour le développement du logiciel, le schéma suivant montre la structure globale des classes.



** la classe CVector est quasiment utilisée par toutes les autres classes*

Descriptions des classes :

CBitmapGL

Permet de dessiner des images ou du texte (2D).

CTga

Charge ou enregistre une image TGA et crée des textures.

CTimerGLUT

Permet de récupérer le temps écoulé et de calculer le nombre d'images par seconde.

CMouse

Enregistre les coordonnées et l'état des boutons de la souris.

CVector

Représente un vecteur ou un point et définie des opérations vectorielles.

CCameraSystem

Permet de gérer un ensemble de caméras.

CSolarSystem

Représente le système solaire.

CObject

Un objet contient une position, un nom et un type.

CPlanet

Représente une planète (et ses déplacements) avec toutes ses caractéristiques.

CSkyBox

Boîte englobant tout le système solaire avec une texture représentant un fond étoilé.

CSpaceShip

Représente le vaisseau (objet 3DS) et permet de le déplacer.

CCamera

Caméra fixe.

CCameraFree

Caméra libre pouvant se déplacer et regarder n'importe où.

CCameraFreePlanet

Caméra pouvant regarder n'importe où mais suivant les déplacement d'une planet.

CCamera3Person

Camera placée derrière le vaisseau.

CParticleSystem

Système permettant de simuler des particules. (classe abstraite)

CReactor

Système de particule très simple permettant de simuler le réacteur du vaisseau.

Modélisation du système solaire :

La bibliothèque OpenGL de base supporte la modélisation et la restitution de points simples, de lignes et de polygones pleins convexes. Pour modéliser les sphères la GLU propose des routines permettant de modéliser des sphères via les équations quadratiques.
Pour dessiner les planètes nous avons donc utilisé la fonction `gluNewQuadric` permettant de créer un objet quadrique. Ensuite il ne reste plus qu'a appeler la `gluSphere` pour dessiner une sphère. Enfin les fonctions `gluQuadricNormals` et `gluQuadricTexture` permettent d'activer la gestion des lumières et des textures sur la planète.

Le fond étoilé est simuler à l'aide d'une boite qui englobe tout le système solaire, celle-ci est texturée à l'aide d'une image noir contenant de petit points blancs (les étoiles). Cette technique très simple à mettre en œuvre permet de faire croire à l'utilisateur qu'il se trouve dans l'espace.

Les textures (des planètes, du fond, du vaisseau) sont chargées à partir d'images au format TGA. La classe `CTga` s'occupe de les charger et de créer les textures correspondantes. De plus, pour éviter les effets visuels désagréables on utilise le mipmapping. Cette technique consiste à partir d'une texture initiale à générer des textures de résolutions décroissantes qui seront plaquées sur l'objet en fonction de la distance à laquelle il se trouve. La fonction utilisé pour effectuer le mipmapping est la suivante : `gluBuild2DMipmaps`.

La gestion des lumières est très simple puisqu'il n'a que deux sources de lumières : l'une permet d'éclairer uniformément le soleil, l'autre est une lumière en provenance de soleil qui éclaire dans toutes les directions (cela permet au planètes de n'être éclairées que du coté exposé au soleil).

Caméras :

Les caméras sont simplement implantés à l'aide de trois vecteurs : la position `m_vPosition`, la position de l'endroit où elle regarde `m_vView` et un vecteur `m_vUpVector` qui correspond au vecteur perpendiculaire au vecteur de vue dirigé vers le haut.
Pour déplacer la caméra (fonction `MoveCamera` et `StrafeCamera`) il suffit juste d'additionner le vecteur de déplacement au vecteurs de `m_vPosition` et `m_vUpVector`.

Trois types de caméras ont été réalisés :
- Caméra Free permet de se déplacer librement dans n'importe quelle direction.
- Caméra Free Planet permet de suivre le mouvement d'une planète.
- Caméra Third Person permet de suivre un objet (dans notre cas le vaisseau) et d'avoir une vue arrière de l'objet.

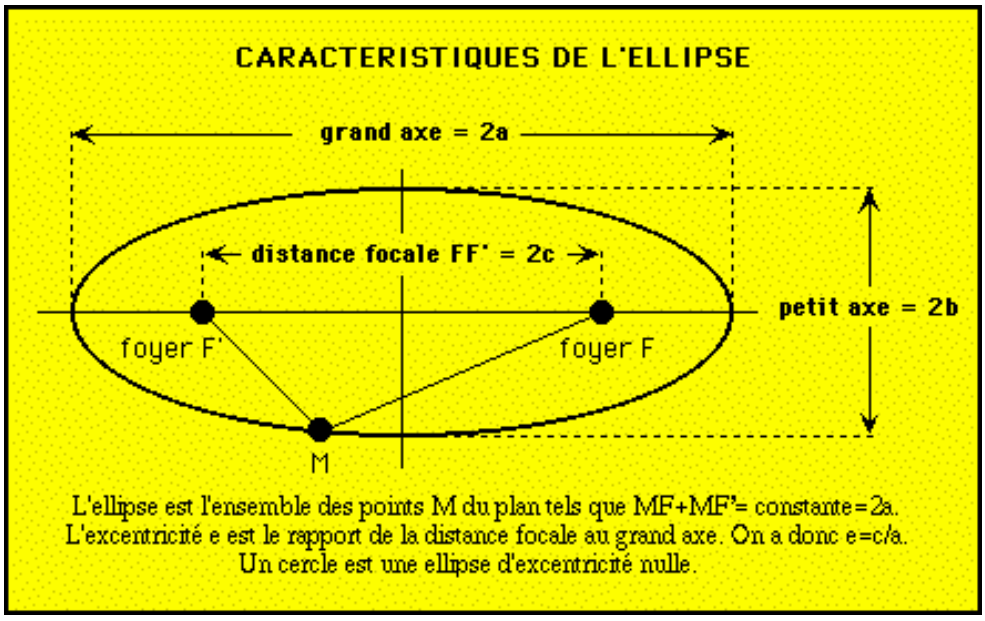
Calcul des trajectoires des planètes :

Pour le mouvement des planètes, on utilise des trajectoires elliptiques. Pour une planète on connaît uniquement l'excentricité de l'ellipse `e`, la distance au soleil `d` et son angle de révolution.
La première étape consiste à calculer le demi grand axe `a`, le demi petit axe `b` ainsi que la demi distance focale `c` grâce au calculs suivants :

```
// m_excentricity = excentricité
// m_distsun = distance au soleil
// m_a = demi grand axe
// m_b = demi petit axe
// m_c demi distance focale

// m_excentricity = m_c / m_a

void CPlanet::SetEllipse()
{
      m_c = (m_excentricity*m_distsun) / (1-m_excentricity);
      m_a = m_c/m_excentricity;
      m_b = sqrt(m_a*m_a - m_c*m_c);
}
```



Ensuite il ne reste plus qu'à faire varier l'angle de révolution de la planète, et de recalculer ses nouvelles positions grâce au deux fonctions suivantes :

```
// Retourne la nouvelle position X de la planète après une rotation
// d'un certain angle autour du soleil
float CPlanet::GetNewX(float angle)
{
	float newX = 0.0;
	// on soustrait m_c car changement de repère, le soleil est le
	// centre et non le centre de l'ellipse (ça change rien pour Y)
	newX = m_a * cos(angle)- m_c;
	return newX * 0.5;
}

// Retourne la nouvelle position Y de la planète après une rotation
// d'un certain angle autour du soleil
float CPlanet::GetNewY(float angle)
{
	float newY = 0.0;
	newY = m_b * sin(angle);
	return newY * 0.5;
}
```

Enfin pour dessiner les trajectoires, on stocke simplement un tableau (d'une taille en rapport avec la dimension de l'ellipse) pour chaque planète contenant des points de l'ellipse. Il ne reste plus qu'à relier ces points pour afficher la trajectoire.

Vaisseau spatial :

La modélisation du vaisseau spatiale est faite en chargeant un fichier 3D Studio grâce au loader fournit dans les fichiers `3dsloader.h` et `3dsloader.cpp`. On utilise également une texture TGA contenant un certain degré alpha de transparence permettant de simuler la poussée du réacteur. Enfin un cube rouge entour le vaisseau ce qui permet le repérer plus aisément.

Pour les déplacements on fixe une vitesse au vaisseau, celui-ci se déplacera en fonction de cette vitesse dans une direction. Les touches du clavier ainsi que la souris permettent de changer la direction et la vitesse (la vitesse est constante et il n'y a pas de facteur d'accélération).

Sélection des planètes :

Lorsque l'utilisateur utilise le bouton droit de la souris sur une planète ou sur une trajectoire, la planète correspondante est sélectionnée.
OpenGL prévoit un mécanisme de sélection qui indique automatiquement l'objet qu'un utilisateur à spécifier avec le pointeur de la souris.
Ce système utilise les fonctions suivantes pour dessiner les objets :

```
glInitNames, glPushName
```

et les fonctions suivantes pour retrouver un objet sélectionné :

```
glSelectBuffer, glRenderMode(GL_SELECT), gluPickMatrix
```

Le mode zoom dessine une planète en gros plan, tournante sur elle-même et avec toutes ses caractéristiques ainsi que son symbole affichées.

Code source

Cette section présente la définition de quelques classes utilisées dans SolarSys.

CObject :

```
// Types des objets
enum TypeObject
{
	OBJECT,
	CAMERA,
	CAMERA_FREE,
	CAMERA_FREE_PLANET,
	CAMERA_3PERSON,
	PLANET,
	SPACESHIP
};

// ##########################
// Classe CObject (abstraite)
// ##########################
class CObject
{
public:
	// Constructeur (identifiant + nom)
	CObject(id_t id, char * name);

	// Destructeur
	~CObject();

	// Dessine l'objet
	virtual void Draw();

	// Change le nom
	void SetName(char * name);

	// Retourne le nom
	char * GetName();

	// Change l'identifiant
	void SetID(id_t id);

	// Retourne l'identifiant
	id_t GetID();

	// Change le type
	void SetType(int type);

	// Retourne le type
	int GetType();

	// Retourne la position
	CVector GetPosition();

	// Change la position
	void SetPosition(float x, float y, float z);

protected:
	// Nom
	char * m_name;
	// Identifiant
	id_t m_id;
	// Type (OBJECT, CAMERA, ...)
	int m_type;
	// Position
	CVector m_vPosition;
};
```

CCamera :

```
// ##############
// Classe CCamera
// ##############
class CCamera : public CObject {

public:
	// Constructeur
	CCamera(id_t id, char * name);

	// Destructeur
	~CCamera();

	// Retourne la position de la cible de la camera
	CVector GetView()		{ return m_vView;		}

	// "Up" vecteur ( en general = (0,0,1) )
	CVector GetUpVector() { return m_vUpVector; }

	// Positionne la camera (modifie les vecteurs position, cible et "up")
	void PositionCamera(float positionX, float positionY, float positionZ,
			    float viewX,     float viewY,     float viewZ,
			    float upVectorX, float upVectorY, float upVectorZ);

	// Utilise la fonction gluLookAt() pour dire a OpenGL ou regarder
	void Look();

	// Deplace la camera ainsi que la cible vers la gauche
	// ou la droite suivant la valeur de speed
	void virtual StrafeCamera(float speed);

	// Deplace la camera ainsi que la cible vers l'avant ou
	// l'arriere suivant la valeur de speed
	void virtual MoveCamera(float speed);

	// Change l'angle de vue de la camera en fonction des deplacements de la souris
	// x,y correspondent a la position courante de la souris
	// lx,ly correspondent a la derniere position de la souris (avant le deplacement)
	void virtual SetViewByMouse(int x, int y, int lx, int ly, float speed);

	// Deplace la camera ainsi que la cible en fonction de la touche "key"
	void virtual SetViewByKeyboard(int key, float speed);

	// Dessine la camera
	void virtual Draw();

	// Mise a jour de la position de la camera
	void virtual Update();

protected:
	// Position de la cible de la camera
	CVector m_vView;

	// "Up" vecteur de la camera
	CVector m_vUpVector;
};
```

CPlanet :

```
// ##############
// Classe CPlanet
// ##############
class CPlanet : public CObject
{
public:
	// Objet quadric utilise pour dessiner les spheres
	static GLUquadricObj *QUADRIC;

	// Timer global
	static CTimerGLUT *TIMER_GLUT;
```

```
        // Classe permettant d'effectuer des dessins bitmaps
        static CBitmapGL *BITMAP_GL;

        // Facteur permettant de regler la vitesse de rotation
        static float ROTSPEED_FACTOR;

        // Facteur permettant de regler la vitesse de revolution
        static float REVSPEED_FACTOR;

        // Affiche les ellipses representant les orbites des planetes
        static bool ELLIPSES;

        // Constructeur
        CPlanet(id_t id, char *name);

        // Destructeur
        ~CPlanet();

        // Dessine la planete
        void Draw();

        // Dessine le soleil (eclairage different)
        void DrawSun();

        // Dessine la planete en mode Zoom avec ces informations
        void DrawZoom();

        // Serie de fonctions Get et Set pour les différents attributs de la classe
        // . . . . .

        // Initialise les donnees pour calculer l'ellipse decrite autour du soleil
        void SetEllipse();

        // Retourne la nouvelle position X de la planete apres une rotation
        // dans certain angle autour du soleil
        float GetNewX(float angle);

        // Retourne la nouvelle position Y de la planete apres une rotation
        // dans certain angle autour du soleil
        float GetNewY(float angle);

private:
        // Initialise la texture
        void InitTexture(char * name_img);

        // Image representant le symbole de la planete
        CTga *m_symb;
        // Identifiant de la texture
        GLuint m_texture;
        // Angle de rotation de la planete sur son axe
        float m_rotationAngle;
        // Vitesse de rotation autour de son axe
        float m_rotationSpeed;
        // Vitesse de revolution autour du soleil
        float m_revolutionSpeed;
        // Demi grand axe de l'ellipse décrite par la planète
        float m_a;
        // Demi petit axe de l'ellipse décrite par la planète
        float m_b;
        // Distance centre de l'ellipse-foyer (soleil)
        float m_c;
        // Angle du mouvement elliptique de la planète
        float m_revolutionAngle;
        // Coordonnees X,Y pour dessiner l'ellipse
        float *m_ellipseX;
        float *m_ellipseY;

        // Rayon
        float m_radius;
        // Distance par rapport au soleil
        float m_distsun;
        // Inclinaison de l'equateur sur l'orbite
        float m_incline;
        // Masse
        float m_mass;
        // Excentricite de l'orbite
```

```
        float m_excentricity;
        // Duree de la rotation sidérale en minutes
        int m_rotationLength;
        // Duree de la revolution sidérale en jours
        float m_revolutionLength;
        // Densite
        float m_density;
        // Gravite
        float m_gravity;
        // Inclinaison de l'orbite
        float m_orbitIncline;
        // Temperature
        float m_temperature;
        // Constituants atmospheriques
        char m_constituents[20];
        // Nombre de satellites
        int m_nbSatellites;
};
```

CSpaceShip :

```
// ##################
// Classe CSpaceShip
// ##################
class CSpaceShip : public CObject
{
public:
        // Constructeur
        CSpaceShip(id_t id, char * name);

        // Destructeur
        ~CSpaceShip();

        // Dessine le vaisseau
        void Draw();

        // Augmente la vitesse
        void SpeedUp();

        // Diminue la vitesse
        void SpeedDown();

        // Tourne a droite
        void TurnRight(float elapsedTime);

        // Tourne a gauche
        void TurnLeft(float elapsedTime);

        // Tourne vers le haut
        void TurnUp(float elapsedTime);

        // Tourne vers le bas
        void TurnDown(float elapsedTime);

        // Change la direction du vaisseau en fonction des deplacements de la souris
        // x,y correspondent a la position courante de la souris
        // lx,ly correspondent a la derniere position de la souris (avant le deplacement)
        void SetDirectionByMouse(int x, int y, int lx, int ly, float speed);

        // Change la position du vaisseau en fonction de sa direction et de sa vitesse
        void UpdatePosition(float elapsedTime);

        // Retourne la direction
        CVector GetDirection();

        // Change l'acceleration
        void SetAcceleration(float acceleration);

        // Change la vitesse
        void SetVelocity(float velocity);

        // Retourne la vitesse
        float GetVelocity();
```

```
private:
	// Permet de faire tourner le vaisseau d'un angle en radians
	void Rotate(float angle, float x, float y, float z);

	// Direction
	CVector m_vDirection;
	// "Up" vecteur
	CVector m_vUpVector;
	// Vitesse
	float m_velocity;
	// Vitesse minimal
	float m_minVelocity;
	// Vitesse maximal
	float m_maxVelocity;
	// Acceleration
	float m_acceleration;
	// Vitesse de rotation
	float m_rotationSpeed;

	// Angle de rotations pour dessiner le vaisseau
	float m_rotXY;
	float m_rotZ;

	// Vaisseau (points et coordonnees de la texture)
	obj_type m_object;

	// Halo du reacteur
	CReactor *m_reactor;
};
```

CSkyBox :

```
// ###############
// Classe CSkyBox
// ###############
class CSkyBox : public CObject
{
public:
	// Constructeur
	// Largeur, hauteur, longueur, position x y z
	CSkyBox(float width, float height, float length, float x, float y, float z);

	// Destructeur
	~CSkyBox();

	// Initialise la texture en fonction de la face du SkyBox
	// et a partir d'une image TGA
	void InitTexture(int side, CTga *);

	// Creation des listes d'affichage necessaire pour dessiner la SkyBox
	void CreateDisplayLists();

	// Dessine la SkyBox
	void Draw();

private:
	// Largeur
	float m_width;
	// Hauteur
	float m_height;
	// Longueur
	float m_length;

	// Tableau contenant les identifiants des textures
	GLuint m_textures[6];
	// Tableau contenant les textures chargees
	bool m_texturesOk[6];

	// Identifiant de la liste d'affichage
	GLuint m_displayList;
};
```

CSolarSystem :

```
// ###################
// Classe CSolarSystem
// ###################
class CSolarSystem
{
public:
	// Constructeur
	CSolarSystem(CTimerGLUT *timer, CBitmapGL *bitmap);

	// Destructeur
	~CSolarSystem();

	// Dessine le systeme solaire
	void Draw();

	// Retourne le mode d'affichage
	unsigned char GetDrawMode();

	// Change le mode d'affichage
	void SetDrawMode(unsigned char mode);

	// Retourne la planete selectionnee
	unsigned int GetSelectedPlanet();

	// Change la planet selectionnee
	void SetSelectedPlanet(unsigned int planet);

	// Retrouve l'identifiant de la planete selectionne en fonction
	// de la position de la souris
	int RetrievePlanetID(int x, int y);

	// Retourne une planete du syteme solaire
	CPlanet * GetPlanet(unsigned int planet);

private:
	// Initialise une planete
CPlanet * InitPlanet( id_t id, char *name, float diameter, float distsun, int rotLength,
                      float incline, float massearth, float excentricity, float revLength,
                      float density, float orbitIncline, float gravity, float temperature,
                                     char *constituents, int nbSatellites );

	// Tableau des planetes
	CPlanet **m_planets;

	// Timer global
	CTimerGLUT *m_timer;

	// Pour dessiner des bitmaps
	CBitmapGL *m_bitmap;

	// Mode d'affichage
	unsigned char m_drawMode;

	// Planete selectionnee
	unsigned int m_selectedPlanet;

	// Position de la lumiere en provenance du soleil
	GLfloat m_sunLightPosition[4];
};
```

CTga :

```
// ############
// Classe CTga
// ############
class CTga
{
public:
	// Constructeur
	CTga();
```

```
        // Destructeur
        ~CTga();

        // Fonction lisant un fichier TGA 8, 16, 24 ou 32 bits (non compresse)
        int Load(char *filename);

        // Change les valeurs alpha des couleurs r,g,b en fonction du parametre cmp
        // Si cmp > 0 on change les couleurs superieures a rgb
        // Si cmp < 0 on change les couleurs inferieures a rgb
        // Si cmp == 0 on change les couleurs egales a rgb
        void AlphaForColor(unsigned char r, unsigned char g, unsigned char b, unsigned char
alpha, char cmp);

        // Fonction créant une image TGA dans le fichier filename
        int static WriteTGAFile(char *filename, short int width, short int height, unsigned
char bpp, unsigned char* imageData);

        // Retourne le format OpenGL de l'image (GL_RGB, GL_RGBA, GL_LUMINANCE)
        GLenum GetTextureFormat();

        // Retourne le nombre de pixels en largeur de l'image
        short int GetWidth();

        // Retourne le nombre de pixels en hauteur de l'image
        short int GetHeight();

        // Retourne le nombre de bits par pixel (RGB = 24, RGBA = 32, Niveaux de gris = 8)
        unsigned char GetBitCount();

        // Retourne le tableau contenant les pixels de l'image
        unsigned char * GetData();

        // Fonction permettant de tester si un entier est une puissance de 2
        bool static CheckSize(int x);

        // Creation d'une texture a partir de l'image courante
        int CreateTexture(GLuint id, bool mipmapping, GLint textureWrap = GL_REPEAT);

private:
        // Format OpenGL de l'image (GL_RGB, GL_RGBA, GL_LUMINANCE)
        GLenum m_texFormat;

        // Nombre de pixels en largeur de l'image
        short int m_width;

        // Nombre de pixels en hauteur de l'image
        short int m_height;

        // Nombre de bits par pixel (RGB = 24, RGBA = 32, Niveaux de gris = 8)
        unsigned char m_bitCount;

        // Tableau contenant les pixels de l'image
        unsigned char *m_data;
};
```

Manuel utilisateur

Manuel définissant les touches utilisables du clavier ainsi que les boutons et les mouvements de la souris : (ce manuel est accessible dans le logiciel en utilisant la touche `h`)

Caméra free :

Permet de se déplacer librement dans le système solaire.

`left` ou `q` :	déplace la caméra vers la gauche
`right` ou `d` :	déplace la caméra vers la droite
`up` ou `z` :	déplace la caméra vers l'avant
`down` ou `s` :	déplace la caméra vers l'arrière
`bouton gauche` de la souris :	effectue une rotation de la caméra
`v` :	augmente la vitesse de déplacement de la caméra
`b` :	diminue la vitesse de déplacement de la caméra

Vaisseau spatial :

La caméra est placer derrière le vaisseau et le clavier et la souris permettent de le diriger.

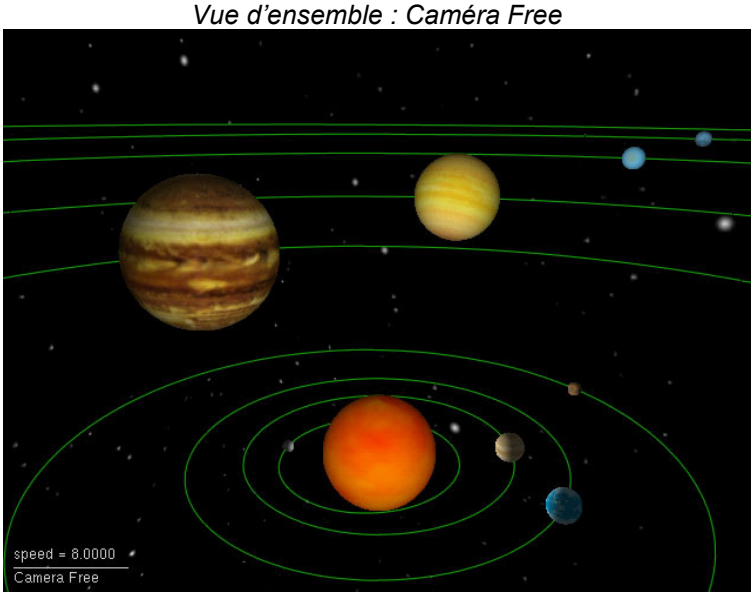
`j` :	fait tourner le vaisseau à gauche
`l` :	fait tourner le vaisseau à droite
`k` :	fait monter le tête du vaisseau
`i` :	fait descendre le tête du vaisseau
`bouton gauche` de la souris :	permet de faire tourner le vaisseau
`w` :	augmente la vitesse du vaisseau
`x` :	diminue la vitesse du vaisseau

Autres commandes :

`bouton droit` de la souris :	sélectionne une planète (mode zoom) ou retourne en vue camera/vaisseau
`+`, `=` :	en mode zoom permet de passer à la planète suivante
`c` :	change la caméra
`e` :	affiche les échelles et les vitesses utilisées
`o`, `p` :	augmente/diminue les vitesses de rotation des planètes
`t` :	affiche/cache les trajectoires des planètes
`f` :	affiche/cache le nombre d'image par seconde
`m` :	mode plein ou mode filaire
`echap` :	quitter

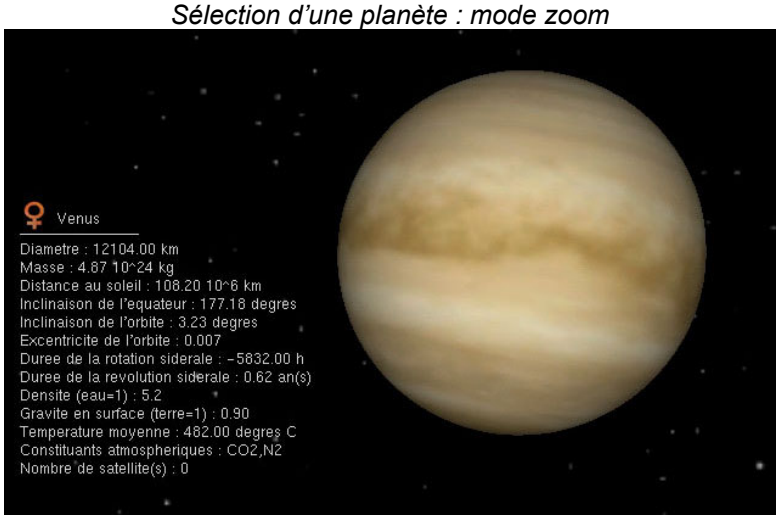
Photos d'écran

Vue d'ensemble : Caméra Free



Vaisseau spatial

Sélection d'une planète : mode zoom



Conclusion

Le projet fut très enrichissant puisqu'il nous a permis de développer une application complète en suivant quelques parties majeures de la création d'une application :

- Assimilation des techniques de programmation C++ et OpenGL.
- Gestion du projet et partage des tâches.
- Recherche sur le domaine visé.
- Analyse pour une bonne structuration du projet.
- Développement de l'application.
- Phase de tests.
- Rédaction du manuel et du rapport.

Beaucoup d'améliorations sont bien entendu possible, comme par exemple :

- Mouvement des planètes suivant les lois de Képler.
- Simulation plus réaliste des déplacements du vaisseau spatial.
- Ajout des satellites autour des planètes.
- Ajout de la ceinture d'astéroïdes.
- … etc …

Nous sommes dans l'ensemble plutôt satisfait de notre travail, les options demandées ont été pratiquement implémentées de la façon demandée. De plus nous avons ajouté quelques notre touche personnelle au projet en ajoutant quelques options supplémentaires.

Informations personnelles

Projet réalisé par VOURIOT Thierry et FUCH Sébastien du 10 avril 2003 au 19 mai 2003 dans le cadre de la maîtrise d'informatique de l'université Louis Pasteur de Strasbourg (67).

Nous joindre :

fuchsseb2@wanadoo.fr
thierry.vouriot@free.fr

Nos sites perso :

http://thierry.vouriot.free.fr
http://perso.wanadoo.fr/fuchs.sebastien