MASTER EN SCIENCES INFORMATIQUES

Simulateur astronomique tridimensionnel destiné à la modélisation d'un système solaire et des éclipses

Mascart, Christian

Award date:
1996

Awarding institution:
Universite de Namur

Année Académique 1995-1996.

Mascart Christian

Mémoire de fin d'études

Simulateur astronomique tridimensionnel destiné à la modélisation d'un système solaire et des éclipses.

Promoteur: Monsieur le Professeur Claude Cherton.

Mémoire rédigé et présenté en vue de l'obtention du grade de Licencié en Informatique.

F.N.D.P. Namur.

Titre 1 : Table des Matières:

Titre 2 : Introduction.

Chapitre 1 : La création

Depuis l'aube de l'humanité, l'homme tourne son regard vers le ciel avec admiration et humilité. Cette attitude marque le passage de l'animal à l'être humain et l'éveil d'une certaine spiritualité.

Incapable d'accéder aux réalités qui y évoluent, il y place des êtres bienveillants ou maléfiques qui gouvernent sa destinée. Le monde est coupé en deux:

Il y a: l'accessible et l'inaccessible,
les hommes et les dieux,
la vie et la mort,
la Terre et le Ciel.

Cette fascination marque le début de la race humaine, son détachement de l'animal. Le fait de se poser une question au sujet de quelque chose dont on n'a pas besoin et de se mettre en quête de la réponse marque le début d'un processus historique.

A la différence de l'animal, qui n'a pas d'histoire et obéit à des instincts immuables, l'homme progresse, évolue et apprend pour améliorer sans cesse son ordinaire. Cette évolution se remarque également dans l'astronomie.

Chapitre 2 : L'évolution

Section 1 : Conceptions du monde:

L'évolution de l'homme se traduit dans tous les domaines, également dans sa façon de percevoir le ciel.

§1 : Egocentrisme

Au départ, la Terre est un disque plat, j'habite au milieu. Tout autour, il y a l'océan, ceinturé d'une immense chute vers les enfers.

§2 : Géocentrisme

La Terre est une sphère portée par trois éléphants posés sur le dos d'une tortue[1]

Puis, elle est une sphère qui flotte dans l'*éther* autour de laquelle tourne tout autre astre. Les anciens avaient déjà une idée de la sphéricité de la Terre en voyant une ombre de forme ronde s'avancer sur la Lune lors des éclipses de Lune.

§3 : Héliocentrisme

Copernic démontre que le Soleil est au centre de l'univers, les planètes tournent autour de lui[2]:

[1]Mythologie hindoue

[2]Galilée y avait déjà pensé longtemps avant lui, mais sa contradiction avec la Bible a failli lui coûter très cher. *Et pourtant, elle tourne !*

§4 : Relativité

Einstein démontre que tout est relatif, il n'y a pas de point de repère.

Georges Lemaître, qui lui est contemporain, dit que tout provient d'une explosion originelle, le Big Bang. Depuis lors, l'univers est en expansion, toutes les choses s'éloignent les unes des autres.

D'autres théories plus récentes proposent un univers périodique, tantôt en expansion, tantôt en contraction.

Section 2 : Accès au ciel

§1 : L'oeil

Tout premier instrument astronomique, l'oeil fut longtemps le seul moyen d'explorer le ciel. L'astronomie ne fit pas de bien grands progrès durant des millénaires. Chaque civilisation avait sa vision du ciel et ses croyances. Elles n'évoluaient gère.

Dans sa soif de connaissances, l'homme a inventé de nouveaux moyens d'acquérir des informations au sujet de ce qui le fascine.

§2 : Le téléscope

Galilée inventa la première lunette astronomique et la pointa vers le ciel. Il découvrit des montagnes sur la Lune, quatre lunes autour de Jupiter et deux étonnantes formes en croissant autour de Saturne, qui s'avéreront plus tard être un immense anneau ne touchant l'astre en aucun point.

§3 : La radio

L'oeil, avec ou sans téléscope, ne nous permet d'observer que l'étroite bande du spectre à laquelle nous sommes sensibles. Lors de l'invention des récepteurs radio, nous nous sommes rendu compte que le ciel recelait des 'émetteurs radio' extrêmement bavards. Sur Terre, la radio fut inventée comme un moyen de communication. L'idée que le ciel puisse émettre relançait la fascination de l'homme pour l'astronomie, croyant qu'il y avait au loin des êtres qui voulaient aussi communiquer.

Actuellement, des téléscopes spéciaux, dits *radiotéléscopes*, explorent le ciel dans des bandes de fréquences autres que le visible. Ils nous permettent de découvrir le ciel sous un aspect totalement inconnu et ont considérablement augmenté notre volume de connaissances, mais nous n'avons à ce jour détecté aucun message intelligent d'origine céleste.

§4 : La fusée

Les moyens précédents sont passifs. Nous ne pouvons connaître du ciel que ce qui nous arrive (rayonnements et météorites). La fusée nous permet de nous arracher à l'attraction terrestre. Il est alors possible de lancer des sondes vers les astres que nous voulons observer. La fusée est un moyen d'observation actif. L'homme sur la Lune et les sondes Voyager aux confins du système solaire en sont un bel exemple.

§5 : L'informatique

Il peut sembler étonnant de considérer l'ordinateur comme moyen d'accès à l'espace. Un ordinateur ne peut traiter que les informations dont il dispose. Celles-ci doivent figurer en mémoire ou lui être accessibles par ses périphériques.

1 ° Découvrir

Certes, un ordinateur peut être placé en orbite ou lancé vers un astre bien précis, mais il est sourd et aveugle sans ses périphériques qui restent, quant à eux, les seuls vrais moyens d'exploration de

l'espace. Cependant, les informations générées par de tels périphériques de plus en plus sophistiqués sont tellement complexes que, sans de puissants algorithmes de manipulation, elles ne sont plus directement compréhensibles. Il faudra par exemple transcrire en une carte en fausses couleurs les indication d'un magnétomètre ou d'un gravimètre, pour *voir* le champ magnétique ou les anomalies du champ gravitationnel d'un astre.

Nous voyons ici resurgir un des paradigmes de la téléinformatique: L'encapsulation des possibilités d'un arsenal logiciel en une couche qui offre ses services à la couche supérieure. Ce paradigme me semble plus poussé en astronomie qu'en téléinformatique, car il modifie fondamentalement la nature même des données selon le **modèle** qu'on s'en fait.

Reprenons l'exemple du champ gravitationnel, il n'a au départ rien à voir avec un phénomène optique qu'est la couleur. La gravité agit sur le poids des choses et non leur aspect visuel. Pourtant, l'observateur d'un champ gravitationnel observe une carte colorée et non un champ gravitationnel lui-même. L'informatique modifie fondamentalement notre manière de percevoir les choses à un point tel que nous ne percevons plus ce qui est, mais ce que l'ordinateur nous présente.

2 ° Modéliser

Les engins d'exploration, terrestres ou spatiaux, permettent de formidables moissons de données. Les découvrir et les amasser est une chose, les organiser, les rapprocher et les synthétiser en est une autre. Des découvertes peuvent être faites par soumission de données existantes à de nouveaux algorithmes.

A un niveau individuel, l'ordinateur permet d'offrir un autre point de vue sur des réalités qui nous échappent à cause de leurs dimensions, de leur éloignement et aussi à cause de notre position sur un astre en mouvement que nous croyons fixe.

On ne parlera donc plus de découvertes, mais d'apprentissage.

Chapitre 3 : Objet du mémoire

Section 1 : Modélisation du système solaire

Le logiciel qui accompagne ce mémoire est précisément un logiciel d'apprentissage. Il modélise un système solaire dans un espace virtuel tridimensionnel et permet de l'observer sous différents angles grâce à une caméra que l'utilisateur pilote. Il peut alors voir tourner les astres et se déplacer parmi eux.

Section 2 : Modélisation des éclipses

L'objet de ce mémoire est aussi de permettre de comprendre le mécanisme des éclipses. Une éclipse est le passage d'un astre dans l'ombre d'un autre. La face éclairée de l'astre éclipsé devient donc totalement ou partiellement sombre. Une éclipse est donc une tache maculant totalement ou partiellement la face éclairée d'un astre. Notre position sur un des astres impliqués dans l'éclipse rend ce phénomène particulièrement difficile à comprendre. Mon logiciel permet de s'abstraire de cette contingence et offre une vision globale du problème.

Section 3 : Aspect personnel

D'un point de vue personnel, ce mémoire a pour objectif de me familiariser avec de nouvelles techniques de programmation. Avant de venir à l'Institut, je programmais en QuickBasic 4.5 tournant sous un environnement MS-DOS. Je connaissais les paradigmes de base:

Séquence d'instructions,
Condition,
Boucle for et while,
Procédure,
Fonction,
Tableau,
Type structuré,
Gestion des disques et des fichiers,
Manipulation de chaînes,
Divers types d'écrans graphiques, sous différentes résolutions, différents niveaux de couleurs, avec une seule ou plusieurs pages-écran,
Programmation de l'imprimante,
Modem: Envoi de caractères, attente et traitement des caractères reçus. (Programmation événementielle rudimentaire),
etc...

De nouvelles techniques se développaient, sous des environnements graphiques. Mis à part la maigre programmation de mon modem, j'ignorait tout des messages et événements.

J'ignorais également les objets et de leurs concepts (héritage, encapsulation , etc...), ainsi que des unités de compilation telles que Pascal les propose. Elles sont beaucoup plus propres et structurées que les modules *fourre-tout* du QuickBasic. Leur découpe en une interface et une implémentation les rend beaucoup plus abordables et performantes, ainsi que les relations *uses*.

J'ai également découvert la manière de concevoir une interface distincte du programme, sous Resource Workshop, en prélude à la programmation événementielle.

Titre 3 : Éléments constitutifs d'une éclipse.

Chapitre 1 : Le plan orbital

Application au Système Soleil-Terre-Lune:

La Terre tourne autour du Soleil selon une orbite elliptique dont le Soleil occupe un des foyers. Cette orbite est inscrite dans un plan appelé l'*écliptique*. Elle est ainsi dénommée car c'est dans ce plan que se produisent les éclipses.

La Lune décrit également une ellipse autour de la Terre dans un autre plan appelé le *plan de l'orbite lunaire*.

Chapitre 2 : Le cône d'ombre

Le cône d'ombre est l'ensemble de tous les points de l'espace où l'astre occultant recouvre totalement le Soleil.

Le Soleil est de loin le plus grand astre du système solaire. Son diamètre surpasse celui de tous les autres astres. Il en résulte que chaque astre projette, dans une direction opposée à celle du Soleil, un cône d'ombre de longueur finie.

La longueur du cône d'ombre Lc dépend de trois facteurs:

1) Le rayon du Soleil: Rs
2) Le rayon de l'astre: Ra
3) La distance entre le Soleil et l'astre: Dsa

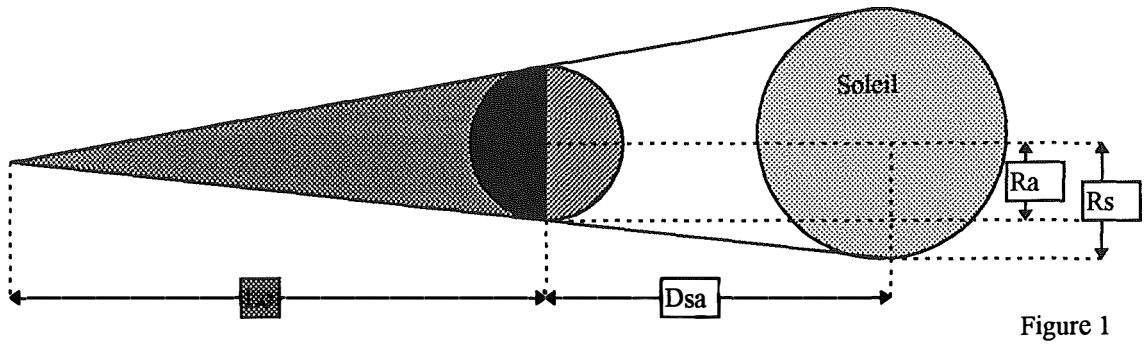

Figure 1

$$LC = ((Dsa / (Rs - Ra)) * Rs) - Dsa$$

Un observateur placé à l'intérieur du cône d'ombre voit le Soleil disparaître totalement derrière l'astre occultant.

Chapitre 3 : Le cône de pénombre

Le cône de pénombre est l'ensemble de tous les points de l'espace où l'astre occultant recouvre partiellement ou totalement le Soleil. Il inclut donc le cône d'ombre.

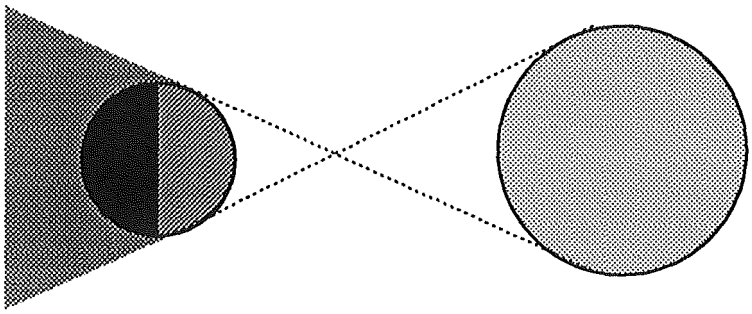

Figure 2

Chapitre 4 : La ligne des noeuds

Le plan de l'orbite lunaire n'est pas confondu avec l'écliptique. Il fait avec celui-ci un angle de 5°. L'orbite lunaire comporte donc deux points remarquables:

Le noeud ascendant,
Le noeud descendant.

Il s'agit des deux points de percée de l'écliptique par la Lune.

Le noeud ascendant est le point que la Lune franchit lorsqu'elle passe de la face sud à la face nord de l'écliptique.

Le noeud descendant est le point que la Lune franchit lorsqu'elle passe de la face nord à la face sud de l'écliptique.

La droite qui les relie est l'intersection des deux plans orbitaux. Elle effectue une lente rotation rétrograde en 18.5 ans par rapport aux étoiles.

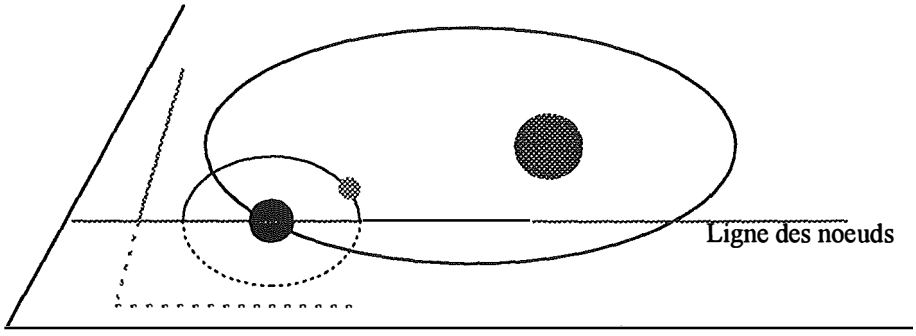

Figure 3

Chapitre 5 : Conditions d'une éclipse

Section 1 : Condition d'alignement

Pour qu'un astre puisse projeter son ombre sur un autre, il faut qu'une partie au moins de l'astre occulté soit comprise dans le cône d'ombre de l'astre occultant.

Cet événement a lieu à la double condition que:

La Lune soit proche du plan de l'écliptique, c'est à dire d'un des deux noeuds.
La Lune passe entre la Terre et le Soleil (Terre éclipsée) ou l'inverse (Lune éclipsée).

§1 : Tolérance

L'alignement du Soleil, de la Terre et de la Lune n'est jamais parfait, une tolérance due au diamètre des astres est admise. En ce qui concerne les éclipses de Lune, il y a éclipse partielle lorsqu'à la pleine ou nouvelle Lune, le noeud est à moins de 12° du Soleil. La tolérance tombe à 5.5° pour une éclipse lunaire totale. Une tolérance existe également pour les éclipses de Soleil.

§2 : Conséquences de la tolérance

Les figures qui suivent illustrent les différentes positions du noeud[3] par rapport à l'alignement Terre-Soleil. Le grand cercle représente le cône d'ombre de la Terre tronqué à la distance Terre-Lune, le petit représente la Lune. La ligne horizontale représente le plan de l'écliptique, la ligne oblique le plan de l'orbite lunaire.

1 ° Pour les éclipses de Lune

a) Eclipse totale maximale

L'alignement impeccable n'est pas toujours requis. Il a pour résultat de voir la Lune traverser le cône d'ombre selon son diamètre. La totalité est maximale.

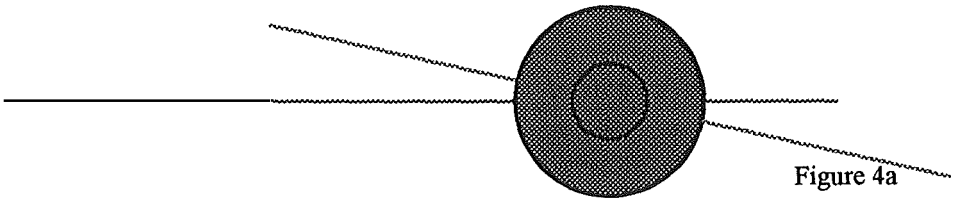

Figure 4a

b) Eclipse totale minimale

Voici l'écart maximal que peut prendre le noeud pour qu'il y ait éclipse totale. Dans ce cas, la lune sort du cône d'ombre de la Terre immédiatement après avoir fini d'y entrer. Il s'agit d'une éclipse totale dont la totalité a une durée nulle. Elle se caractérise par un pôle (sud ici) lunaire extrêmement brillant. Cela est dû à l'atmosphère terrestre. Il convient de préciser qu'à cause de cette dernière, le cône d'ombre terrestre est assez flou.

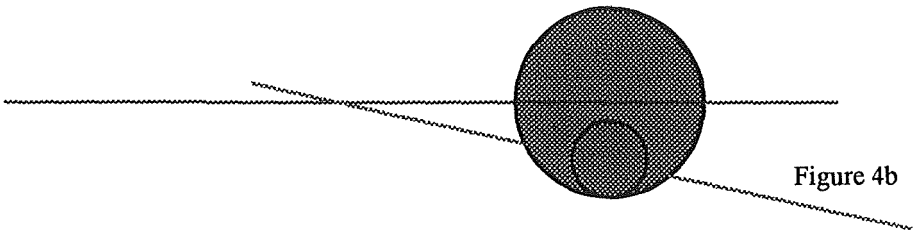

Figure 4b

c) Eclipse partielle minimale

Voici l'écart maximal que peut prendre le noeud pour qu'il y ait éclipse partielle. Dans ce cas, nous avons une éclipse partielle qui ne dure pas. Il y a tout au plus un léger assombrissement au pôle au moment de la pleine Lune. Le reste de la Lune est quant à lui normal.

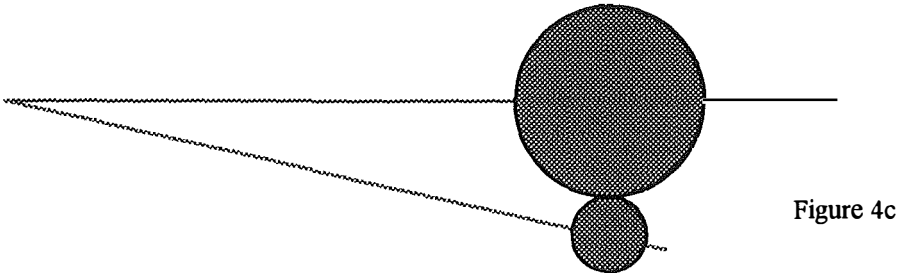

Figure 4c

[3]Il s'agit ici du noeud ascendant.

2 ° Pour les éclipses de Soleil

a) Définition d'une éclipse

Vu notre position sur l'astre occulté, Il importe ici d'introduire la différence entre une éclipse et une occultation:

Eclipse

Une éclipse est le passage d'un astre dans l'ombre d'un autre.

Une éclipse est un phénomène en soi, indépendant de tout observateur. Dans le cadre d'une éclipse que nous appelons *de Soleil*, c'est en réalité la Terre qui s'éclipse. Une éclipse de Soleil dure environ deux heures. Il s'agit là du temps écoulé entre le moment où la Terre touche le cône d'ombre de la Lune, quelque part au lever du Soleil, et le moment où l'ombre de la Lune quitte la Terre, quelque part au coucher du Soleil.

Occultation

Une occultation est le passage d'un astre derrière un autre.

Ici intervient la notion d'**observateur**, induite par le mot *derrière*.[4] C'est l'observateur qui définit ce qui est visible et ce qui est occulté. L'occultation du Soleil par la Lune, en un lieu terrestre donné, ne dure que quelques minutes. Pour un avion[5] volant dans l'ombre de la Lune à la même vitesse que lui, L'occultation peut durer beaucoup plus longtemps.

La distinction entre une éclipse partielle et une éclipse totale de Soleil n'a aucun sens. Pour un observateur situé sur la ligne de centralité[6], l'occultation du Soleil sera totale. Pour son collègue situé un peu plus au nord ou au sud, elle ne sera que partielle. Pour un autre observateur encore plus éloigné, il n'y aura pas d'occultation du tout.

b) Incidence de l'alignement.

De l'alignement correct ou non de la ligne des noeuds dépendra la **latitude**[7] de la ligne de centralité. Un alignement impeccable donnera une occultation du Soleil à l'équateur. Un mauvais alignement donnera une occultation très peu visible uniquement au pôle.

Section 2 : Condition de distance

La distance entre les deux astres, au moment de l'alignement, doit être inférieure à la longueur du cône d'ombre de l'astre occultant. Si cette distance est supérieure à la longueur du cône d'ombre, l'astre occultant sera incapable de masquer l'entièreté du Soleil. L'astre occulté ne passera que dans la pénombre de l'astre occultant.

---

[4]Les tables d'éphémérides publient également des cartes d'*occultations rasantes*. Une occultation rasante est le passage de la Lune devant une étoile de sorte que le pôle de la lune frôle cette étoile. On peut donc la voir s'allumer et s'éteindre au gré des montagnes et des vallées lunaires. La carte mentionne la ligne sur laquelle il faut se placer, à quelques kilomètres près, pour observer le phénomène.

[5]La NASA a mis au point un avion-observatoire muni d'un téléscope prévu à cet effet.

[6]La ligne de centralité est la ligne reprenant toutes les positions successives de l'intersection de l'axe du cône d'ombre lunaire avec la surface terrestre.

[7]Sous réserve de l'inclinaison de l'axe des pôles. Je suppose ici une Terre dont le plan équatorial est confondu avec l'écliptique. En réalité, il est incliné de 23° 27'.

L'orbite d'un astre n'est pas un cercle, mais une ellipse dont l'astre central occupe un des foyers. De cette particularité, nous pouvons déduire deux points remarquables de l'orbite d'un astre:

Le Périastre,
L'Apoastre.

Il s'agit des intersections entre le grand axe de l'orbite et l'orbite elle-même. Le périastre est l'intersection située du côté de l'astre central. Ces notions sont marginales pour la survenance d'une éclipse. Le système Soleil-Terre-Lune comporte cependant une particularité remarquable:

Le cône d'ombre lunaire a une longueur très proche du rayon moyen de l'orbite lunaire.

Lorsque la Lune est proche de l'apogée, la pointe de son cône d'ombre est incapable d'atteindre la surface de la Terre. Une éclipse totale est alors impossible, même en cas d'alignement des trois astres. La Lune est incapable de masquer totalement le Soleil qui, au plus fort de d'éclipse, reste visible sous forme d'un anneau lumineux autour de la Lune. C'est une éclipse annulaire.

Lune à l'apogée:

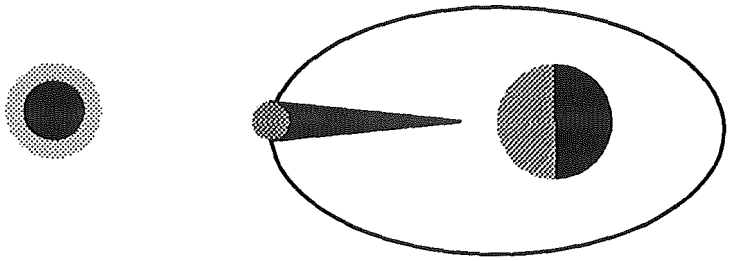

Figure 5a

Lune au périgée:

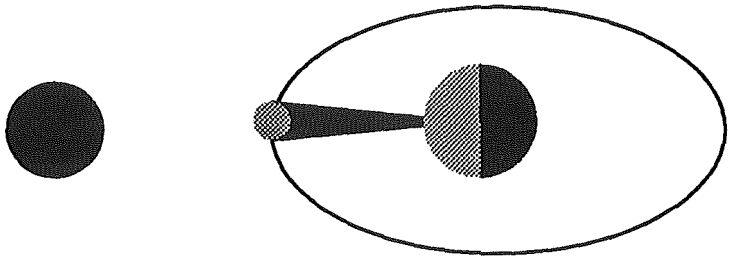

Figure 5b

Titre 4 : La programmation orientée objets

Le simulateur est écrit en Borland Pascal 7.0 orienté objets pour Windows. Son examen consistera en une analyse des différents objets qui le composent ainsi que leurs interactions.

L'acte de programmer consiste à décrire complètement la manière de résoudre un problème afin qu'un autre acteur (l'ordinateur) puisse le résoudre ultérieurement en l'absence du programmeur. Cette description doit avoir lieu dans un langage clair et univoque afin d'être compris par la machine ou, ce qui est plus généralement le cas, par un traducteur, dit *compilateur*, qui transcrira le texte descriptif du problème en un langage dit *machine*, compréhensible par l'ordinateur

Lorsqu'un problème est très complexe, il arrive souvent que le programmeur n'arrive plus à en avoir simultanément une vue d'ensemble et une approche détaillée à une ligne de code près. Les concepteurs de compilateurs ont alors élaboré des langages permettant de découper le problème en plusieurs sous-problèmes plus simples interagissant de manière strictement définie avec le monde extérieur.

Chapitre 1 : Un précurseur: La procédure

Lorsqu'il décrit un problème, le programmeur chemine rarement d'un pas monotone vers une solution de laquelle chaque ligne de code le rapproche. Si tel était le cas, le programme serait une suite tranquille d'instructions séquencées de manière telle qu'en arrivant avec le problème entier sur la première, il est résolu lorsque la dernière est atteinte et que le programme se termine.

Il arrive au contraire fréquemment qu'un problème complexe puisse être **divisé** en problèmes plus simples, et que plusieurs lignes de code soient écrites en vue de résoudre un sous-problème bien précis.

A la notion de sous-problème correspond celle de sous-programme, autrement appelé *procédure*. La procédure est une structure au sein de laquelle le programmeur va regrouper les lignes de code tendant à résoudre un sous-problème bien précis de son problème global.

Écrire une procédure, c'est non seulement résoudre un sous-problème, mais tout d'abord, bien le **poser**. Le programmeur devra, lors de l'écriture de sa procédure, décrire la manière dont le sous-problème interagit avec le problème global.

Section 1 : Déclaration

Cette étape de la conception d'une procédure consiste à décrire la manière dont elle interagit avec le monde extérieur. Il s'agit de lui dire ce qu'elle doit savoir avant de résoudre le sous-problème qui lui est confié et ce qu'elle doit dire une fois qu'elle l'a résolu.

Concrètement, le programmeur dote sa procédure de **paramètres**. Il s'agit de variables que le programmeur confie à la procédure. Certaines lui sont confiées à titre purement informatif, d'autres lui sont confiées afin qu'elle les modifie, pour y placer par exemple les résultats de ses calculs.

Il s'agit respectivement des paramètres passés **par valeur** et des paramètres passés **par adresse**. Dans le premier cas, une simple copie du renseignement est donné à la procédure. Si elle le modifie, cela ne présente aucune importance, car la copie est détruite lorsque la procédure a terminé son travail.

Dans le second cas, c'est l'adresse mémoire du renseignement qui est donnée à la procédure. Procédure et programme appelant vont donc, pendant le temps de vie de la procédure, partager une zone mémoire commune que le programme retrouvera éventuellement modifiée après terminaison de la procédure. Le paramètre par adresse ne contient parfois aucune information avant exécution de la procédure. Il existe simplement pour que la procédure y place quelque chose.

Section 2 : Définition

Une fois les paramètres déclarés, le programmeur doit rédiger le corps même de la procédure, c'est à dire décrire les étapes de résolution du sous-problème comme il l'aurait fait pour le problème principal, à une échelle moindre cette fois ci. Au terme de cette description, dite *définition*, le sous-problème doit être résolu.

Section 3 : Approche récursive

Si le sous-problème est encore trop volumineux que pour être compris d'emblée, le programmeur peut recommencer l'approche précédente: Déclarer des sous-procédures à sa procédure et ainsi de suite jusqu'à la trivialité. Un problème est dit *trivial* lorsque sa solution est évidente. Il n'est alors plus nécessaire de le découper.

§1 : exemple:

Pour savoir si un astre, dont je connais les coordonnées rectangulaires (X, Y, Z), est dans mon champ visuel, je dois en déterminer les coordonnées polaires. Pour en déterminer l'angle d'azimut, je dois calculer un angle décrivant le ratio de ses coordonnées X et Y.

Pour déterminer la hauteur, je vais calculer l'angle décrivant le ratio entre d'une part sa coordonnée Z et d'autre part la distance entre sa projection sur le plan XY et l'origine. Ces deux problèmes étant identiques, je vais créer une procédure qui va me donner l'angle d'un point connu par son ordonnée et son abscisse. J'appellerai deux fois cette procédure, une fois pour en obtenir l'azimut, une fois pour en obtenir la hauteur.

La procédure de calcul des coordonnées polaires d'un astre comportera une sous-procédure de détermination d'un angle, qu'elle appellera deux fois. Nous voyons apparaître ici un autre avantage de la programmation procédurale: L'**économie de lignes de code**.

Une fois que la procédure de calcul des coordonnées polaires tridimensionnelles de l'astre a terminé son travail, elle confie ses résultats à la procédure de test d'inclusion qui l'a appelée. Elle saura alors si l'astre est inclus dans son champ visuel et si oui, où.

Chapitre 2 : Définition d'un objet

Section 1 : Faiblesses de la procédure

Ayant compris que diviser un problème permet de mieux le résoudre, les programmeurs ont eu besoin d'un outil permettant d'accentuer plus loin encore cette découpe du programme. La programmation orientée objets est une réponse à ce besoin.

La procédure restait malgré tout une réponse fort artificielle au besoin de découper un problème. Une fois qu'elle a terminé son travail, elle disparaît de la scène et oublie tout ce qu'elle a fait. Les seules traces de son passage sont les variables qu'elle a modifié. Rédiger une procédure oblige donc le programmeur à garder un oeil sur le contexte dans lequel elle sera employée, c'est à dire sur le problème global.

L'amnésie de la procédure oblige donc à conserver dans un espace commun les variables nécessaires à résoudre le problème global. Or, certaines variables ne servent qu'à régler un aspect très particulier du problème. Aspect qui n'intéresse pas les autres sous-problèmes.

Section 2 : Forces de l'objet

L'idée qui s'est imposée est de regrouper, sous une notion commune qui restait à inventer, les procédures et les variables qu'elles manipulent. Le tout formerait une sorte d'être intelligent qui connaît des choses et sait ce qu'il doit en faire. Il recevrait ses ordres de l'extérieur, les exécuterait dans la mesure du possible et pourrait informer celui qui lui a donné l'ordre du suivi de son ordre.

Cet être, c'est l'**objet**. Tout d'abord, il s'agit d'une structure collective, un peu comme un *record*, qui comporte autant de champs qu'il doit connaître de choses. Mais bien plus, l'objet contient aussi, outre ses champs, des procédures, désormais appelées **méthodes**, aptes à manipuler ces champs au gré des ordres que l'objet reçoit selon les algorithmes qu'elles contiennent.

§1 : L'interface

Incarnation d'un sous-problème, l'objet est appelé à communiquer avec le monde extérieur pour participer, dans le mesure de ses possibilités, à la solution du problème global. Il doit donc offrir une certaine prise au monde extérieur par laquelle il peut être manipulé. L'interface d'un objet est la manière dont il peut être envisagé de l'extérieur: Ce qu'il requiert et ce qu'il propose.

Il en va de même d'une voiture, qui reçoit les commandes de son conducteur par ses pédales, son volant et son levier de vitesse ainsi que différents contacts électriques (Clé, levier de commande des phares et des clignoteurs, etc...)

Une date peut être envisagée comme un curseur sur une ligne du temps: on peut le faire avancer, reculer, lui assigner une position donnée, l'interroger sur sa position, ...

§2 : L'implémentation

Il s'agit ici de la partie privée d'un objet. L'implémentation contient la manière dont l'objet résout le problème qui lui est posé. Elle n'est pas visible de l'extérieur. Elle est 'ce qu'il y a sous le capot' de l'objet.

Par exemple, une voiture à traction avant offre les mêmes organes de commande qu'une propulsion arrière, mais la manière d'arriver au même résultat (avancer sur la route) est tout à fait différente: La première fait tourner ses roues avant et la seconde exerce son effort au moyen des roues arrières.

Section 3 : Pensée 'orientée objets'

§1 : Pour le concepteur

L'orientation objets n'est pas qu'une manière de programmer, elle est aussi une manière de voir les choses. Elle consiste tout d'abord à isoler une pièce de son contexte en repérant ses interactions avec ledit contexte (l'interface), puis à y consacrer toute son attention afin de la réaliser de la manière la plus performante possible (l'implémentation).

Une fois isolé de son contexte, l'objet est le seul souci du programmeur. Il doit tout mettre en oeuvre pour que l'objet puisse réaliser efficacement son travail. Idéalement, la façon de concevoir l'objet ne devrait avoir aucune incidence sur les services qu'il offre.

Prenons par exemple le point situé dans un espace virtuel. Peu importe qu'il mémorise sa position en coordonnées polaires ou rectangulaires, l'essentiel est qu'il puisse donner l'une ou l'autre selon les besoins. Il peut, par exemple, mémoriser ses coordonnées rectangulaires dans trois champs (un pour chaque dimension) et disposer d'une fonction de conversion offrant ses coordonnées polaires. L'inverse est également possible, c'est au programmeur de voir quelle est la forme la plus souvent sollicitée par le contexte, car la fonction de conversion prend du temps alors que donner un champ par l'intermédiaire d'une fonction qui n'effectue aucun calcul est (presque) immédiat. Si les deux formes sont souvent sollicitées pour un point qui n'est pas trop souvent déplacé, le point peut mémoriser ses coordonnées sous les deux formes. A charge pour lui de bien veiller à la cohérence entre les deux en les mettant à jour toutes les deux lors de chaque déplacement du point dans l'espace.

§2 : Pour l'utilisateur

L'orientation objets est aussi une simplification pour l'utilisateur qui doit élaborer un programme utilisant les objets qu'un autre a créé, ou qu'il a créé plus tôt et dont il ne se souvient plus des détails de réalisation. L'essentiel pour l'utilisateur est de connaître **l'interface** d'un objet afin de savoir ce qu'il offre et nécessite. Peu lui importe de savoir comment il fonctionne.

Ainsi, un électricien a besoin de savoir de quelle type de source de courant il dispose pour réaliser son installation:

Monophasée,
Triphasée 220V,
Triphasée 380V + Neutre.

Mais peu lui importe que l'objet qui lui fournisse cette tension soit un transformateur, un compteur électrique ou un alternateur. L'essentiel est qu'il obtienne les fils aux tensions voulues de son objet 'Source de courant électrique'.

Chapitre 3 : Interaction entre objets

Tout comme la procédure interagit avec l'extérieur par ses paramètres, l'objet interagit avec le monde extérieur par son interface, qui est elle-même un ensemble de méthodes éventuellement munies de paramètres.

La question est alors de savoir quel est ce monde extérieur. Nous avons vu qu'une approche récursive était possible en matière de procédures, il en va de même avec les objets, mais de manière plus complexe.

Section 1 : Dans les méthodes

Pour résoudre un problème bien précis dans une méthode donnée, un objet peut avoir besoin d'un autre objet. Par exemple, lorsqu'un point situé dans un espace tridimensionnel doit résoudre ce problème bien précis qui est de calculer ses coordonnées polaires, il va devoir créer plusieurs objets angles qui savent mieux que lui résoudre les problèmes trigonométriques.

Section 2 : Dans les champs

Un objet peut lui-même contenir en lui plusieurs autres objets. Si une voiture est une réalité complexe, un moteur est complexe également. On ne va donc pas dire que la voiture a une culasse, des pistons, des bielles, des soupapes, des bougies comme on dirait qu'elle a des roues, un volant et un châssis, on va dire qu'elle a *un moteur*, des roues un volant et un châssis. En prenant soin de revenir sur le moteur plus tard.

L'astre obscur (planète, satellite) a également un angle de phase qui est un objet angle.
Cette relation signifie qu'un objet en **a** un autre.

Section 3 : Par héritage

Un objet peut être une spécialisation d'un type plus simple dont il a toutes les caractéristiques.
Prenons par exemple l'armoire: Elle a une hauteur, une largeur et une profondeur. Elle a également une porte. Elle sait comment calculer son volume et sait répondre à un ordre d'ouverture/fermeture de la porte. Elle peut également contenir d'autres objets.

Pour construire un frigo, il suffit de dire que c'est une armoire, puis d'écrire les lignes de codes propres au frigo: Il a un *compresseur*, (éventuellement défini ailleurs), pour produire du froid, il a une température de fonctionnement, une consommation électrique, etc...
Cependant, dès que l'on a dit qu'il s'agit d'une armoire, il n'est plus nécessaire d'expliquer qu'il a une hauteur, une largeur et une profondeur, ni qu'il sait comment calculer son volume et répondre à un ordre d'ouverture/fermeture de la porte.

L'héritage permet de créer un objet commun à plusieurs autres. Par exemple, l'armoire reprend les caractéristiques communes au frigo et au coffre-fort. Pour créer un coffre-fort, je prend une armoire déjà toute faite, et je lui met une serrure qui mémorisera le code secret, ce qui aura pour conséquence de modifier la méthode d'ouverture de la porte.

Cette relation signifie qu'un objet en **est** un autre.

Définir un type armoire est intéressant, même si je n'ai besoin que de frigos et de coffres-forts, car l'objet armoire reprendra les caractéristiques communes au frigo et au coffre-fort, Il en résulte également une économie de lignes de code.

C'est pour cette raison (Cfr. infra), que j'ai créé un objet *astre* dont héritent l'astre brillant et l'astre obscur. L'astre constitue un tronc commun à tous les astres du système solaire, brillants (Soleil) ou obscurs (Planètes et satellites).

Titre 5 : Modélisation logicielle d'une éclipse:

Le simulateur tridimensionnel accompagnant ce mémoire offre un espace virtuel dans lequel gravitent des astres. Cet espace contient également une caméra qui permet à l'utilisateur d'en prendre connaissance.

Le simulateur modélise bien plus qu'une éclipse, il modélise tout un système solaire paramétrable dans lequel se produisent ou peuvent se produire des éclipses.

Chapitre 1 : L'objet Angle

L'angle est fondamental, tant en astronomie qu'en géométrie tridimensionnelle, pour travailler avec des coordonnées polaires. Il est donc logique qu'un objet lui soit consacré, afin d'en faciliter la manipulation.

Le type d'angle dont j'ai besoin est un angle destiné à recevoir les résultats d'un calcul qui lui est extérieur. En d'autres termes, un angle donné ne devra jamais générer une information qu'il recevra itérativement en retour sur un grand nombre de cycles.

Concrètement, le positionnement d'un astre auquel l'angle est destiné ne se fera pas par incrémentation journalière d'un angle donné à un angle de phase inhérent à l'astre. Cette technique donne lieu à une dérive si l'angle n'est pas une variable d'une précision suffisante.

L'angle de phase d'un astre reçoit chaque jour une valeur 'fraîche' en provenance d'un autre algorithme. Peu importe donc qu'il se produise un léger arrondi lors de l'affectation.

Initialement, lors de la construction de mon angle, j'avais prévu de mémoriser sa valeur en degrés car cette unité est plus répandue dans le public. Néanmoins, les fonctions offertes par Pascal utilisent et offrent des angles en radians.

C'est donc en radians que l'angle mémorise sa valeur, la saisit et la donne. On peut cependant lui donner ou demander une valeur en degrés.

Un angle peut également être additionné ou soustrait d'un autre angle. Dans ce cas, une méthode interne à l'angle veille à maintenir la valeur de l'angle dans un intervalle compris entre 0 et 2 Pi.
Dans le même ordre d'idées, lorsqu'on communique à un angle une valeur excédant 360° ou 2 Pi, ou inférieure à zéro, il la ramène dans un rang acceptable grâce à une méthode privée.

Un angle peut également dire s'il en inclut un autre. Sa réponse est booléenne. Pour qu'un angle A inclue un angle B, il faut que B, ou 2 Pi-B si B>Pi soit inférieur à A (qui est quant à lui toujours inférieur à Pi, puisqu'il s'agit de la moitié du champ de vision de la caméra). Cette méthode est expressément dévolue à la caméra (cfr. infra) qui doit tester la présence d'un point, dont elle connaît les coordonnées polaires, dans son champ visuel. Une méthode plus simple, mais plus onéreuse en temps de calcul, serait de comparer les cosinus. Un angle est inclus dans un autre si son cosinus est plus grand ou égal à celui de l'autre.

Ainsi, 30° est inclus dans 40°.
350° est inclus dans 11°
Tous les angles sont inclus dans 180°.

Un angle peut également donner sa proportion par rapport à un autre angle dont il est requis qu'il soit non nul. Cela permettra à la caméra de savoir où afficher un astre par rapport à son champ visuel (Cfr. infra).

Chapitre 2 : L'objet Date:

La date est le moteur du simulateur. C'est en fonction d'elle que se positionnent les astres. Une date est un point sur une ligne du temps. Elle se déplace dans un univers unidimensionnel. La date est définie au jour près. Le simulateur ne gère pas les sous-multiples du jour (l'heure, la minute et la seconde.)

La ligne du temps sur laquelle elle se déplace a pour bornes le premier janvier de l'an zéro[8] et le 31 décembre 9999. La meilleure modélisation mathématique en est un sous-ensemble de N des nombres naturels.

```
TDate = object (TObject)
        constructor init;
        procedure ajuster (j_donne : INTEGER;
                           m_donne : INTEGER;
                           a_donne : INTEGER);
        procedure Avancer;
        procedure Reculer;
        procedure AujourdHui;
        function numero : LONGINT;
        function humaine : STRING;

        private
         jour, mois, annee     : INTEGER;
         numerodate     : LONGINT;
         function bissextile (an: INTEGER) :BOOLEAN;
       end;
```

Section 1 : Modélisation:

Une date doit pouvoir se donner sous deux aspects:

Un nombre naturel pour les calculs,

Une chaîne de caractère au format JJ-MM-AAAA pour les affichages.

L'objet date comporte donc deux types de champs:

1) Un entier long: NUMERODATE, mémorisant la date sous forme d'une position le long de la ligne du temps.
2) Trois entiers courts mémorisant la date sous forme d'un jour, d'un mois et d'une année.

Il est conçu de sorte que la cohérence entre les deux champs reste constante. Cette cohérence est assurée par la fonction ajuster, demain et hier (Cfr. infra)

Je n'ai pas implémenté de fonction réciproque. Une date se manipulera donc par ses champs JMA dont il sera immédiatement déduit le numéro.

Section 2 : Interface

Outre l'initialisation qui la construit, la date peut recevoir quatre messages:

§1 : Avancer

La date incrémente son champ 'Jour' de 1. Si la nouvelle valeur excède 28 alors s'enclenche l'algorithme de décision de passage au mois suivant. Celui-ci consulte un tableau le renseignant sur le

[8]Évitons ici une querelle d'historiens pour qui l'an zéro n'existe pas. Lorsque le Christ est né fut a proclamé l'an 1 de l'ère chrétienne. L'an 1 succédait donc directement à l'an -1 sans qu'il n'y ait d'an 0.

nombre maximal de jours de chaque mois et décide du passage ou non au mois suivant, et éventuellement à l'année suivante. Elle incrémente également le champ NumeroDate

```
NombreJours : array (1..12) of INTEGER = (31, 28, 31 ,30, 31, 30, 31, 31, 30, 31, 30,
31);
```

§2 : Reculer

Cette méthode a un comportement similaire à la précédente: Le champ 'Jour' est décrémenté de 1. S'il vaut zéro, elle décrémente le mois de 1, et passe éventuellement à l'année précédente. Elle consulte ensuite le même tableau pour prendre connaissance de la nouvelle valeur du champ 'Jour'. Elle décrémente également le champ NuméroDate.

1 ° Février et les années bissextiles:

Les deux méthodes précédentes se heurtent immanquablement au problème des années bissextiles. Elles se poseront respectivement la question de savoir quel est le lendemain du 28 février et la veille du 1er mars.

La seconde case du tableau *NombreJours* n'est jamais consultée. Lorsque le problème se pose, une fonction privée[9] *Bissextile* reçoit l'année par paramètre et répond par un booléen est bissextile ou non.

Bissextile procède par séparation de cas selon le schéma suivant: L'alternative de gauche signifie OUI, celle de droite NON.

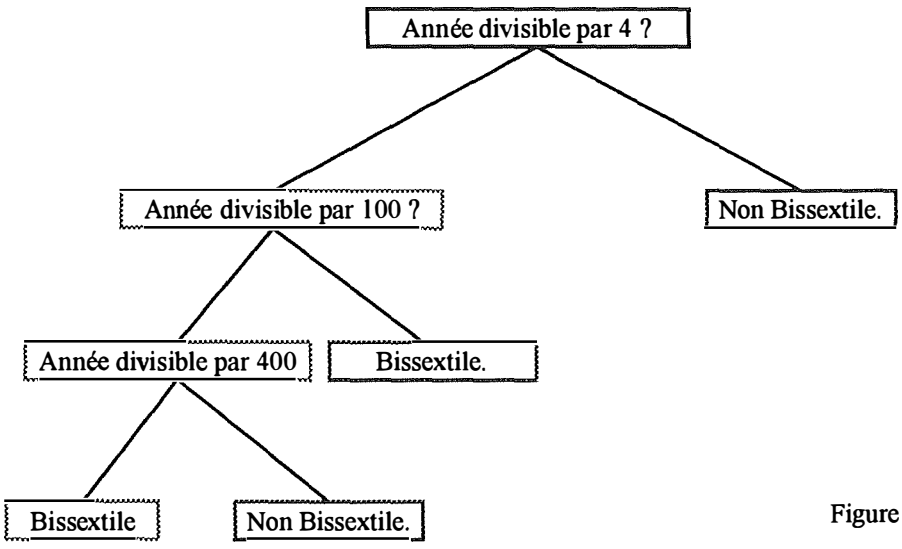

Figure 6

Toutes les années dont l'expression numérale est divisible par 4 sont bissextiles (1988, 1992, 1996). Cependant, les années de siècles ne sont pas bissextiles (1700, 1800, 1900), sauf celles dont les deux premiers chiffres forment un nombre divisible par 4 (1600, 2000, 2400, ...)

2 ° Bouclage:

L'appel répété de chacune de ces deux méthodes risque de conduire la date à une extrémité de la ligne du temps. Il a donc été prévu que:

[9]C'est à dire interne à l'objet date et ne pouvant être accédée de l'extérieur.

Le lendemain du 31-12-9999 serait le 01-01-0000.
La veille du 01-01-0000 serait le 31-12-9999.

L'utilisateur en est informé par un message à l'écran. C'est la seule fonction d'interface de l'objet Date. En cas de bouclage, la cohésion du champ NumeroDate est assurée. L'objet connaît les valeurs des bornes de la ligne du temps.

§3 : Aujourd'hui

Cette méthode consulte le *setup* de la machine grâce à une fonction offerte par Pascal et ajuste la date à cette valeur. Elle est d'office appelée par le constructeur TDate.Init afin qu'une date nouvellement construite ait d'office une valeur valide.

§4 : Ajuster à (Jour, mois, année)

Cette méthode a pour rôle de maintenir la cohérence entre les deux formats de date:

JMA et nombre naturel.

Elle est d'office appelée par les autres méthodes chaque fois que la date change. Elle peut aussi être appelée de l'extérieur de l'objet date, par exemple par un objet d'interface qui voudrait ajuster la date à une valeur donnée.

Si les valeurs auxquelles on voudrait ajuster la date sont invalides, cette méthode détecte l'erreur et place -1 dans le champ NumeroDate de la date.

Une date est erronée à l'une des conditions suivantes:

Année < 0
Année > 9999
Mois < 1
Mois > 12
Jour < 1
Jour > Maximum admissible pour le mois, tel que déterminé précédemment.

§5 : Numéro

Cette méthode est une fonction par laquelle la date donne son numéro au monde extérieur. Elle n'effectue aucun calcul mais se contente de divulguer un champ privé. afin de permettre à d'autres objets qui se servent de la date d'effectuer des calculs.

§6 : Humaine

Cette fonction rend une chaîne de longueur constante de dix octets contenant la date au format:

JJ-MM-AAAA

Elle permet à des objets d'interface de divulguer la date sous un format compréhensible pour l'homme.

Chapitre 3 : L'objet 3D-Point:

Cet objet reprend les caractéristiques de base de tout ce qui est appelé à évoluer dans un espace tridimensionnel.

Section 1 : Méthodes de positionnement:

Sa fonction première est de mémoriser sa position dans un espace tridimensionnel selon ses coordonnées rectangulaires. (X, Y, Z) Il comporte donc trois champs privés mémorisant respectivement l'ordonnée, l'abscisse et l'élévation. Trois méthodes de lecture permettant d'en prendre connaissance et trois méthodes d'écriture permettant de les modifier.

En outre, il offre des méthodes permettant de résoudre certains problèmes de ses héritiers liés à leur localisation spatiale:

Section 2 : Distance

Il peut calculer la distance qui le sépare de l'origine ou d'un autre 3D-Point. Il utilise pour cela le théorème de Pythagore:

Le carré de l'hypoténuse C d'un triangle rectangle est égal à la somme des carrés des deux côtés A et B de l'angle droit.

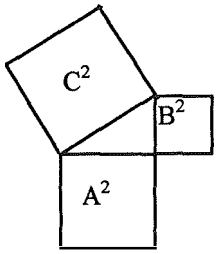

Figure 7

§1 : Distance de l'origine:

Un 3D-Point calcule la distance qui le sépare de l'origine en calculant la racine carrée de la somme des carrés de ses trois coordonnées

§2 : Distance d'un autre 3D-Point:

Pour ne pas implémenter une seconde fois le théorème de Pythagore en tenant compte cette fois-ci des différences entre coordonnées, je relativise[10] l'autre 3D-Point par rapport à *self*[11], puis je lui demande sa distance depuis l'origine.

Section 3 : Coordonnées polaires

Un 3D-Point peut aussi être manipulé polairement, tant en lecture qu'en écriture. Les coordonnées polaires consistent en deux angles:

L'azimut
La hauteur

et une distance le séparant de l'origine.

---

[10] Cfr. Infra: Coordonnées relatives.

[11] Ce terme, en programmation orientée objets, signifie: « L'objet dont il est actuellement question, celui que l'on traite. »

§1 : Principe

Les coordonnées polaires ne sont pas mémorisées. Lors du positionnement d'un 3D-Point en coordonnées polaires, celui-ci doit calculer ses coordonnées rectangulaires et les stocker dans ses champs.

L'angle de hausse lui permet de calculer sa coordonnée Z:

Z := Sin (Hausse) * distance

L'angle de hausse lui permet également de calculer, à titre de variable intermédiaire, la distance qui sépare l'origine de sa projection sur le plan horizontal XY.

Projection := Cos (Hausse) * distance

Cette variable lui permettra de calculer ses coordonnées X et Y:

X := Cos (Azimut) * Projection
Y := Sin (Azimut) * Projection

Lorsqu'on demande à un 3D-Point ses coordonnées polaires il doit les calculer sur base de ses coordonnées rectangulaires selon un principe inverse à celui exposé ci-avant.

Section 4 : Coordonnées relatives

Un 3D-Point peut également donner ses coordonnées non par rapport à l'origine, mais par rapport à un autre 3D-Point. Il peut également s'*additionner* à un autre 3D-Point par addition respective de ses trois coordonnées.

Section 5 : Affichage

§1 : Caméra

Les héritiers des 3D-Points sont destinés à s'afficher à l'écran. Un dialogue devra donc se nouer entre la caméra (Cfr. Infra) et les points à afficher.

La caméra communique ses caractéristiques au point qu'elle envisage d'afficher. Si celui-ci lui est dans son champ visuel, il calcule à quelle coordonnée en 2D il doit apparaître à l'écran.

Cette méthode d'affichage est volontairement incomplète. Il ne s'agit ici que de calculer les coordonnées en 2D et de les stocker dans un champ ad hoc de l'objet 3D-Point sans provoquer le moindre effet graphique à l'écran. Cette méthode sera complétée par héritage selon le type de descendant.

§2 : Position spatiale

Un 3D-Point peut aussi afficher sa position spatiale à l'endroit de l'écran qu'on lui indique. Concrètement, il affiche ses coordonnées sous forme d'une chaîne de caractères au format:

(xxxx, yyyy, zzzz)

Il utilise pour cela la fonction TextOut en lui passant par paramètres la chaîne qu'il vient de constituer et la position à laquelle on lui a demandé de s'afficher.

Chapitre 4 : L'objet 3D-Tache

Une tache est un 3D-Point destiné à être placé sur une sphère. Elle connaît sa position et l'adresse de la sphère sur laquelle elle figure lorsqu'on lui communique la position de la caméra (qui, à ce stade, est un 3D-Point), elle dit si elle est à l'avant-plan ou du côté de la sphère opposé à l'utilisateur.

Deux types de taches devront être représentées sur un astre:
- Le terminateur
- L'éclipse.

Le terminateur est un phénomène permanent. Il s'agit de la ligne de démarcation du jour et de la nuit. Il n'y en a qu'un seul par astre.

L'éclipse est un phénomène temporaire. Il peut y en avoir une, plusieurs ou aucune à la surface d'un astre.

Concrètement, le terminateur sera créé par allocation statique alors que l'éclipse (Les éclipses) devront être allouée(s) dynamiquement lors de leur survenance et devront pouvoir être détruites.

Pour les délimiter, deux objets seront créés, ils serviront à en marquer le pourtour.

Section 1 : La tache permanente

La tache permanente est une tache assignée une fois pour toutes à un astre. Elle est construite en même temps que lui et n'est jamais détruite. Elle a pour vocation d'implémenter le terminateur. Après construction, elle peut être déplacée à volonté par assignation de nouvelles valeurs dans ses champs X, Y et Z afin de répondre aux mouvements du terminateur sur l'astre et de l'astre dans l'espace.

Section 2 : La tache temporaire

La tache temporaire est destinée à représenter un phénomène temporaire à la surface d'un astre. Elle peut être créé et détruite, mais ne peut pas être déplacée sur l'astre.

La tache temporaire peut en outre pointer sur une autre tache temporaire suivante.

Elle est typiquement destinée à implémenter l'éclipse. Lorsque l'astre se rend compte qu'il est éclipsé (Cfr. infra.), il créera une tache temporaire pour chaque point de percée d'une arête du cône d'ombre[12] et les enchaînera. L'astre éclipsé ne devra donc concrètement mémoriser que l'adresse de la première tache temporaire de l'éclipse qui le macule.

L'ensemble des taches temporaires, lors d'une éclipse, détermineront une surface polygonale fermée à la surface de l'astre. Cette surface sera peinte en noir pour former l'éclipse proprement dite.

[12]Nous approximons le terminateur, qui est un cercle à la surface d'un astre, par un polygone comportant un nombre fini de côtés. Il en résulte que le cône d'ombre est approximé par une pyramide comportant un nombre fini d'arêtes.

Chapitre 5 : L'objet 3D-ligne

La 3D-Ligne est une ligne qui relie deux 3D-Points dans l'espace virtuel. Elle est destinée à construire les cônes d'ombre. Une 3D-Ligne est constituée de deux champs de type 3D-Point: L'**origine** et l'**extrémité**.

Section 1 : Construction

La 3D-Ligne est un objet dont les instances seront statiques. Elle est assignée une fois pour toutes à un astre et construite en même temps que lui. Il est donc logique qu'on ne connaisse rien de la 3D-Ligne au moment où on la construit. Le constructeur lui assigne donc le 3D-Point (0, 0, 0) à la fois comme origine et comme extrémité.

Section 2 : Positionnement

Une méthode *FixerBouts* comportant deux paramètres de type 3D-Point permet de positionner la 3D-Ligne entre deux points de l'espace. Le premier paramètre sera transposé dans l'origine et le second dans l'extrémité.

Section 3 : Fonctions offertes

La 3D-Ligne peut, par deux méthodes, donner les deux 3D-Points qu'elle relie.

Section 4 : Rôle

La 3D-Ligne est essentiellement un objet destiné à mémoriser des valeurs et non à effectuer de calculs. Elle est passive car les calculs de percée d'une sphère auxquels elle est destinée sont effectués par la sphère. La 3D-Ligne est un objet destiné à percer une sphère.

Chapitre 6 : L'objet sphère

La sphère est un 3D-Point auquel est ajouté un rayon. Elle est destinée à résoudre le problème des éclipses. C'est elle qui détermine si une 3D-Ligne la transperce. Si c'est le cas, elle mémorise le point d'entrée de la 3D-ligne. Le point d'entrée est le point de percée le plus proche de l'origine ou l'unique point de contact si la 3D-ligne est tangente à la sphère.

Section 1 : Affichage

Une sphère ne peut pas s'afficher en tant que telle à l'écran. Mais, à l'instar du 3D-Point, Une partie du travail d'affichage des descendants de la sphère peut déjà être effectué à son niveau.

Fondamentalement, une sphère est un 3D-Point, Elle va donc, par héritage, en fonction de sa position propre et de la caméra, déterminer si elle est à l'écran et, si oui, quelles sont ses coordonnées 2D.

Si elle est affichable, elle va en tant que sphère déterminer son rayon apparent en fonction de son rayon réel et de la distance qui la sépare de la caméra. C'est le 'Rayon en 2D'.

Pour calculer son rayon en 2D, elle va tout d'abord calculer son 'rayon apparent'. Le rayon apparent est l'angle dont la caméra est le sommet, dont une branche passe par chaque extrémité d'un rayon de sphère perpendiculaire à l'axe provenant du centre de la sphère et passant par la caméra.

En sachant que la tangente = sinus / cosinus, on obtient la tangente de cet angle en divisant le rayon de la sphère par la distance entre le centre de la sphère et la caméra.

La fonction Arc Tangente donne ensuite la valeur du rayon apparent.

La longueur en pixels du rayon en 2D est alors obtenue par une simple règle de trois:

1) Quelle est la proportion du rayon en 2D par rapport à l'angle de vision horizontal ?
2) Quelle est la demi-largeur en pixels de la fenêtre d'affichage ?
3) En multipliant la demi-largeur par la proportion, on obtient le rayon en 2D.

On considère la demi-largeur de la fenêtre et non pas sa largeur totale car le rayon est le demi-diamètre.

Chapitre 7 : L'objet astre:

Un astre est une sphère destinée à être positionnée dans l'espace en fonction de la date.

Section 1 : Arborescence et récursivité

Les astres sont organisés en une structure arborescente au sein de laquelle les messages se propagent par des appels récursifs.

Un descendant est relié à son ascendant par la relation *'tourne autour de'*. La racine en est le Soleil.

C'est au niveau de l'objet astre que se gère la propagation des messages par récursivité. Lorsqu'un message doit être adressé à l'ensemble des astres, il suffit de l'adresser au Soleil qui le répercutera à ses satellites s'il en a. A leur tour, les satellites feront de même, et ce jusqu'au bout des branches qui sont les astres sans satellites.

Cette façon de procéder me semble plus pratique que d'adresser itérativement un message à chaque élément d'un tableau balayé par une boucle. En parlant au Soleil, je suis sûr de n'oublier personne.

L'objet astre comporte tout ce qui est commun aux astres brillants et obscurs. C'est un type abstrait, il ne sera jamais instancié. Il mémorise notamment:

Le nom de l'astre,
La couleur de l'astre
Le nombre de ses satellites
L'adresse de chaque satellite
L'adresse de l'astre central autour duquel l'astre gravite.

C'est au niveau de l'astre que s'effectuent les opérations d'affichage consistant à tracer un cercle de couleur désirée.

La localisation du centre de l'astre est héritée, via la sphère, du 3D-Point,
Le calcul du rayon de l'astre est hérité de la sphère.

Au stade de l'astre, il n'y a plus de calcul à effectuer, mais seulement à procéder aux affichages proprement dit, c'est à dire acquérir une plume et une brosse, s'en servir pour dessiner sur le contexte d'affichage et les rendre.

Méthodes récursives:

Positionnement
Se nommer
Donner son adresse

§1 : Positionnement

Lorsqu'un astre reçoit un ordre de positionnement, il se positionnera selon sa méthode propre, puis préviendra tous ses satellites qui feront de même. Au niveau de l'astre, la méthode de positionnement est volontairement incomplète. Elle se limite à propager l'ordre de positionnement dans l'arborescence.

§2 : Se nommer

Un astre peut recevoir l'ordre d'afficher son nom à l'écran. Dans ce cas, le nom sera affiché à la position du centre en 2D de l'astre.

Cet ordre peut être inconditionnel. Dans ce cas, l'astre n'affichera son nom qu'à la seule condition d'être présent à l'écran.

Cet ordre peut être conditionné à la proximité d'un point, défini par ses coordonnées en 2D. dans ce cas, l'astre n'affichera son nom que si le point qu'on lui communique est compris dans le cercle qui le représente. Il utilisera pour cela le théorème de Pythagore pour apprécier la distance entre le point et son centre en 2D et ne réagira favorablement que si cette distance est inférieure à son rayon en 2D.

§3 : Donner son adresse

A l'instar de la méthode 'Se Nommer', un astre donne son adresse si le point qu'on lui communique est compris dans son cercle d'affichage. Dans ce cas, et contrairement à la méthode 'Se Nommer', il cesse de propager le message plus loin dans l'arborescence, afin qu'un seul astre ne puisse réagir à ce message. Un autre astre qui contiendrait également ce point ne réagira donc pas puisqu'il ne sera pas informé de ce message.

Section 2 : L'astre brillant

Cet objet est destiné à instancier le Soleil. Le pointeur sur son astre central est d'office mis à NIL lors de la construction.

Cet objet n'est pas très différent de l'astre. Il est destiné à n'être instancié qu'une fois. Son nom est d'office 'Soleil', quel que soit le nom donné au constructeur.

§1 : Positionnement

Il répond à un ordre de positionnement en mettant d'office ses coordonnées à (0, 0, 0), sans effectuer de calcul. Le Soleil est donc non seulement la racine de l'arborescence, mais également l'origine du repère tridimensionnel dans lequel se situent chaque 3D-Point.

Ensuite, il appelle la méthode ancêtre 'positionnement', qui sert à propager l'ordre de positionnement à tous les satellites du Soleil, c'est à dire les planètes.

Section 3 : L'astre obscur

C'est à ce niveau que sont traitées les spécificités de l'astre obscur:

Il tourne autour d'un autre,
Il comporte un terminateur,
Il comporte un cône d'ombre,
Il peut recevoir l'ombre d'un autre astre.

§1 : Positionnement

Un astre obscur est un astre qui tourne autour d'un autre. La révolution est un phénomène périodique dont l'équation peut se résumer à un point de départ et à un pas.

Le pas est la période de révolution.

Le point de départ est un angle de déphasage permettant de 'mettre l'astre à l'heure'.

Pour se positionner, l'astre va calculer la partie décimale de la division de la date par sa période de révolution, la multiplier par 360° puis y additionner l'angle de déphasage. A ce moment, il connaît son angle de phase.

Le sinus et le cosinus de cet angle multipliés par le rayon orbital permettent de déterminer ses coordonnées X et Y. Le cosinus de cet angle multiplié par le débattement vertical[13] permet de déterminer sa coordonnée Z.

---

[13]Le débattement vertical permet de donner à l'astre une orbite inclinée sur le plan XY. Dans ce modèle simplifié, la ligne des noeuds ne précessionne pas. Il en résulte que l'orbite d'un astre n'est un cercle que si son débattement vertical est nul. Autrement, il s'agit d'une ellipse dont l'astre central est au milieu des foyers. La projection de cette ellipse sur le plan XY est un cercle. Mon modèle ne permet pas d'implémenter des astres dont l'orbite est très inclinée sur l'écliptique.

Ces trois coordonnées doivent ensuite être additionnées à celles de son astre central pour obtenir les coordonnées définitives dans l'espace virtuel[14].

§2 : Terminateur

Le terminateur est la limite du jour et de la nuit. Il correspond à tous les endroits de l'astre obscur où le soleil est à l'horizon, tant pour se lever que pour se coucher. Le terminateur de la Terre est parfois appelé le cercle d'illumination.

Dans notre cas, le terminateur n'est pas un cercle mais un polygone comportant un nombre fini de côtés déterminé par une constante Ngone.

Concrètement, le terminateur est un tableau [1..Ngone] of 3D-Taches permanentes que l'astre doit positionner après s'être positionné lui-même.

Lorsqu'un astre se positionne, il doit aussi positionner chaque point de son terminateur. Pour cela, l'astre va itérer Ngone fois avec un angle progressant par pas de 360° / Ngone

Avant d'entamer la boucle, l'astre obscur va déterminer la position du soleil relativement à l'astre, c'est à dire la position qu'aurait le soleil si l'astre obscur était à l'origine[15]. Il va ensuite déterminer ses coordonnées polaires. Tout ceci se fait par appel de méthodes héritées du 3D-Point.

Une fois dans la boucle, l'astre va placer polairement chaque point du terminateur dont il déterminera l'angle de hausse et l'azimut de la manière suivante:

```
for i := 1 to NGone do
   begin
    Tourne.EcritDegre (i * (360 div NGone));
    HaussePoint.EcritRadian ((pi/2) * sin(Tourne.radian));
    AzimuthPoint.EcritRadian ((pi/2)* cos(Tourne.radian));

    AzimuthPoint.Additionner (AzimuthSoleilRelatif);
    HaussePoint.Additionner (HausseSoleilRelatif);
    Terminateur[i].PlacerPolaire (AzimuthPoint, HaussePoint, Rayon);
    Terminateur[i].Additionner (Self);
                        {La suite concerne les cônes d'ombre}
   end;
```

Dans un premier temps, la 3D-Tache permanente est placée sur un cercle vertical inclus dans la sphère.

Ensuite, elle est déplacée des coordonnées relatives du soleil, afin de faire du cercle vertical un cercle perpendiculaire à l'axe Astre - Soleil.

Finalement, la fiction d'un astre centré sur l'origine est supprimée en additionnant à la 3D-Tache les coordonnées de l'astre obscur.

---

[14]Rappelons ici que le 3D-Point, dont l'astre hérite, peut additionner ses coordonnées à celles d'un autre par la méthode additionner.

[15]Le soleil relatif aura donc pour coordonnées les coordonnées opposées à celles de l'astre. Pour un astre situé en (10, 20, 30) par rapport à un soleil situé en (0, 0, 0), le soleil relatif a pour coordonnées (-10, -20, -30) par rapport à un astre devenu l'origine (0, 0, 0)

§3 : Cône d'ombre

Chaque astre obscur projette derrière lui un cône d'ombre de longueur finie. Avant de calculer son cône d'ombre, l'astre obscur doit déterminer les coordonnées de la pointe de son cône d'ombre. Cette pointe est un 3D-Point mémorisé en tant que champ de l'astre.

Le calcul des coordonnées de la pointe du cône d'ombre est grandement facilité par la position à l'origine (0, 0, 0) du soleil. Étant donné que les trois 3D-Points Soleil, Astre obscur et Pointe d'ombre sont alignés et que le Soleil est à l'origine, les coordonnées de la pointe d'ombre ne sont rien d'autre que les coordonnées de l'astre obscur multipliées par une constante. Cette constante K est déterminée selon la méthode expliquée ci-avant pour le calcul de la longueur du cône d'ombre:

DistanceOmbre := trunc ((DistanceSoleil / (RayonSolaire - Rayon[16])) * (RayonSolaire));

DistanceOmbre est la distance entre la pointe du cône d'ombre et le centre du Soleil.
DistanceSoleil est la distance entre le centre de l'astre obscur et celui du Soleil.

K := DistanceOmbre / DistanceSoleil;

Le cône d'ombre est l'ensemble des 3D-Lignes qui relient chaque point du terminateur à l'extrémité du cône d'ombre.

---

§4 : Éclipses

Chaque 3D-Ligne du cône d'ombre est communiquée au Soleil, dont l'astre obscur connaissait l'adresse pour calculer le positionnement de la pointe de son cône d'ombre. Le Soleil est ici envisagé comme la racine de l'arborescence des astres. Il répercute itérativement le message contenant les coordonnées de la 3D-ligne et l'adresse de l'astre qui la possède à ses descendants. Chaque astre sera donc informé du message.

Une fois en possession du message, l'astre, envisagé en tant que sphère, déterminera s'il est transpercé par cette 3D-Ligne. Ce test se fait après éviction des cas triviaux ci-après, pour éviter des calculs de percée inutiles:

- L'astre destinataire est plus proche du Soleil que l'astre occultant,
- L'astre destinataire est l'astre occultant,
- L'astre destinataire est plus éloigné du Soleil que la pointe du cône d'ombre de l'astre occultant (C'est à dire l'extrémité de la 3D-Ligne.)

S'il est transpercé, il détermine le point d'entrée de la droite dans la sphère, c'est à dire le point le plus proche de l'origine de la 3D-Ligne[17]. Il initialisera une nouvelle 3D-Tache temporaire aux coordonnées de ce point d'entrée. Chaque 3D-Tache pointe sur la suivante, ou sur NIL, afin de constituer une file. Cette file est mise à zéro lors de chaque changement de position de l'astre, en début de méthode 'positionner'. Ensemble, ces 3D-Taches délimitent le pourtour de l'éclipse sur l'astre.

---

[16]Si le rayon du Soleil était inférieur à celui de l'astre, il en résulterait, selon cette formule, une aberration qui consisterait en un cône d'ombre fictif situé du même côté que le soleil. Pour parer à cette éventualité, un test préalable est effectué. Il a pour but de générer une erreur qui plante le programme.
Cette erreur sera ultérieurement décelée beaucoup plus tôt, lors de la construction, lorsque l'utilisateur tentera de créer un astre plus grand que le Soleil.
[17]Ce point peut également être le point de sortie dans le cas rarissime d'une 3D-Ligne tangente à une sphère.

Une fois que l'astre s'est positionné, terminateur et cône d'ombre compris, il appelle la méthode 'positionner' de son ancêtre pour répercuter l'ordre de positionnement à tous ses satellites.

Notez l'importance de donner cet ordre <u>après positionnement</u> et non avant, puisque le satellite a besoin de connaître les coordonnées de son astre central pour se positionner lui-même.

Chapitre 8 : L'objet caméra

Section 1 : Rôle

§1 : En sortie

Dans le monde réel, une caméra est un appareil qui envisage un monde tridimensionnel selon un certain angle et le projette sur une image en 2D.

L'objet caméra est ici une caméra virtuelle qui observe l'espace virtuel dans lequel se meuvent les astres et en déduit une image bidimensionnelle en vue de son affichage sur une fenêtre Windows. Ce n'est pas à proprement parler un objet d'interface homme-machine, car elle ne propose pas d'image à l'utilisateur. Elle est à l'interface entre le monde tridimensionnel et le monde bidimensionnel propre à la fenêtre.

La caméra est l'objet d'interface qui permet à l'utilisateur de prendre connaissance de l'espace virtuel tridimensionnel. Il n'existe à ce jour aucun périphérique d'affichage en 3D[18]. Il est donc impossible à l'utilisateur d'avoir lui-même une vue directe du monde virtuel tridimensionnel.

§2 : En entrée

La caméra permet également à l'utilisateur de changer de point de vue: Soit en se déplaçant, soit en modifiant la direction de son regard. Ici aussi, la caméra n'est pas un objet d'interface homme-machine. Elle n'écoute pas directement l'utilisateur, mais les messages que l'objet interactif qui la manipule lui envoit.

Section 2 : Conception

La caméra a pour but d'incarner l'opérateur dans sa fonction d'observation de l'espace virtuel. Observer un espace virtuel, c'est en faire partie, se définir par rapport à lui. Fondamentalement donc, la caméra est un 3D-Point. Elle est dans l'espace virtuel qu'elle observe.

Observer un espace virtuel, ce n'est pas seulement s'y situer, c'est aussi le voir sous un certain angle. Tout comme un oeil, la caméra regarde dans une certaine direction. Comme un oeil également, la caméra a un certain champ visuel. Elle peut tantôt se focaliser sur un point, tantôt se comporter comme un grand angle.

En définitive, la caméra peut être envisagée comme une **pyramide quadrangulaire** dont le 3D-Point qui la constitue est le sommet. La direction du regard est définie par deux angles:

L'Azimut, qui définit la coordonnée horizontale du regard.
La Hausse, qui définit la coordonnée verticale du regard.

Le champ visuel est lui aussi constitué de deux angles interdépendants. L'un le définit en largeur, l'autre le définit en hauteur.

La largeur du champ visuel est donnée. Sa hauteur est calculée par une règle de trois selon les dimensions de la fenêtre d'affichage[19] afin que la proportion entre les deux angles soit identique à la proportion entre le largeur et la hauteur de la fenêtre d'affichage.

---

[18]On pourrait imaginer une sorte de cube translucide dans lequel certains pixels pourraient changer de couleur en fonction d'impulsions électriques envoyées par de très fins fils invisibles à l'oeil nu..

[19]La caméra n'a aucune emprise sur les dimensions de la fenêtre d'affichage. elle peut juste en prendre connaissance.

Section 3 : Manipulation

Manipuler la caméra consiste en deux type de mouvements:

La déplacer dans l'espace virtuel,
L'orienter.

Les premiers mouvements s'adressent directement au 3D-Point qui constituent la caméra. L'utilisateur dispose de six commandes permettant de déplacer la caméra dans chaque sens selon les trois directions.

Les seconds mouvements sont des mouvements sur place. Ils consistent à manipuler les deux composantes du regard (Azimut et Hausse) dans les deux directions. L'angle d'azimut peut être incrémenté / décrémenté à volonté, il consiste à faire tourner la caméra sur place selon un axe vertical.

Les valeurs que peut prendre l'angle de hausse ont pour limites +90° et -90°. Il consiste à lever / baisser la tête.

Section 4 : Gestion des affichages

Lorsque la caméra doit afficher le contenu de son champ visuel, elle en est informée par l'appel de sa méthode TCamera.Projeter. Elle reçoit cet ordre accompagné:

de l'adresse du Soleil,
du contexte d'affichage,
De l'option d'affichage ou non des cônes d'ombre.

La caméra va alors explorer récursivement l'arborescence des astres[20] et en noter chaque occurrence dans un tableau. Ensuite, elle va trier ce tableau par ordre d'éloignement des astres. Le premier est le plus lointain, le dernier est le plus proche. Une boucle va alors parcourir l'intégralité de ce tableau en ordonnant à chaque astre de s'afficher. Cet ordre est assorti des paramètres de vision de la caméra:

Position en 3D (La caméra est ici réduite à un simple 3D-Point)
Direction du regard (azimut et hausse)
Champ visuel (hauteur et largeur)
Contexte d'affichage (mention de la fenêtre sur laquelle il doit s'afficher)
Option d'affichage ou non des cônes d'ombre

L'astre va alors appeler la méthode d'affichage de son ancêtre: Le 3D-Point[21], avec un paramètre supplémentaire spécifiant au 3D-Point qu'il ne doit s'afficher que s'il est dans le champ visuel.

Certes, il peut sembler évident qu'un 3D-Point ne doive s'afficher que s'il est dans le champ visuel, c'est à dire dans la fenêtre d'affichage que tout programme Windows crée lors de son initialisation. Néanmoins, certains 3D-Points d'ordre secondaire doivent impérativement s'afficher, car ils participent à la construction de graphismes qui leurs sont extrinsèques.

Il s'agit:

De la pointe du cône d'ombre,
De chaque point du terminateur,
De chaque tache d'une éclipse.

[20]Il s'agit ici du type ancêtre des astres brillants et obscurs. Pour afficher un astre, la caméra n'a pas à savoir s'il est brillant ou obscur, c'est à l'astre de régler lui-même son affichage en fonction de ses propres spécificités.
[21]La méthode d'affichage n'a pas été redéfinie au niveau de la sphère, de sorte que l'astre hérite ici directement de son grand-père, le 3D-Point.

Lorsque la caméra demande à un astre de s'afficher, celui-ci teste sa présence dans le champ visuel de la caméra qu'il reçoit par paramètres. Ce test porte sur la présence du centre de l'astre, et non d'un point quelconque de sa surface. Car l'astre est centré sur le 3D-Point dont il hérite. C'est donc le centre qui, par héritage, va effectuer ce test et dire :

S'il est affichable (Réponse booléenne)
Si oui, où sur la fenêtre (La réponse est un TPoint[22].)

L'affichage, même partiel, d'un astre a lieu à la seule et unique condition que son centre soit à l'écran. Dans ce cas, TOUS les autres points qui composent l'astre doivent s'afficher pour tracer les lignes dont ils sont une extrémité.

Les astres auxquels la caméra s'adresse sont soit des astres brillants (il n'y en a qu'un: le Soleil), soit des astres obscurs. Or, elle les envisage comme des astres, sans donc tenir compte de leurs spécificités. C'est donc aux méthodes virtuelles 'afficher' de faire le reste. La méthode 'afficher' de l'astre ancêtre trace un disque dont le rayon est calculé par la méthode 'afficher' de la sphère.

Ensuite, et selon le type d'astre, l'astre brillant trace des rayons solaires, l'astre obscur trace un terminateur et un éventuel cône d'ombre si le paramètre lui demandant de tracer les ombres est vrai.

Ici joue le paramètre d'affichage obligatoire: Faux pour les astres, il sera vrai pour les 3D-Taches qui le maculent et pour le bout du cône d'ombre[23]. Celles-ci doivent donc calculer leurs coordonnées en 2D, même si elles sont hors du champ visuel. J'exploite ici une possibilité de Windows de faire des 'affichages virtuels'. C'est à dire qu'un dessin hors des limites de la fenêtre ne provoque rien, même pas une erreur. Un dessin (par exemple, un trait) qui commence dans la fenêtre et se termine en dehors sera réalisé partiellement pour sa partie comprise dans la fenêtre. Tous les accessoires de l'astre sont donc conditionnés à l'affichage de l'astre.

Pourquoi est-ce le 3D-Point qui calcule son affichage en fonction des caractéristiques de la caméra ?

Une solution a priori logique serait que la caméra regarde les points et leur dise où elle les voit, c'est à dire leur assigne une position à l'écran.

Au contraire, la solution retenue est une caméra qui prévient tout le monde de sa position et de son champ visuel, à charge pour chaque 3D-Point de procéder lui-même aux calculs.

La raison en est que, lorsque la caméra regarde des astres, elle n'en connaît pas les accessoires. Elle explore l'arborescence, trie les astres et les prévient dans l'ordre afin que le plus lointain soit recouvert par le plus proche. Par contre, elle ignore de quoi sont faits les astres qu'elle affiche, s'ils contiennent d'autres points à afficher également. Un dialogue devrait donc se nouer entre la caméra et l'astre:

1) La caméra voit l'astre et lui donne une position à l'écran.
2) La caméra demande à l'astre s'il a d'autres points à placer à l'écran également
3) L'astre lui répond affirmativement et lui envoit un autre 3D-Point.
4) La caméra en calcule la position à l'écran et en informe l'astre propriétaire.
5) Retour en (2) jusqu'à ce qu'il n'y ait plus d'astre à afficher.

Ici, la caméra 'se donne' à l'astre qui, à son tour, en répercute les caractéristiques à d'autres 3D-Taches qui le composent.

[22]Le TPoint est un type structuré offert par Pascal, qui comprend deux champs: X et Y, destiné à afficher un point à l'écran.
[23]L'affichage ou non du cône d'ombre ne dépend donc que de la demande d'affichage des ombres par l'utilisateur, et non de la présence de la pointe du cône d'ombre à l'écran.

Titre 6 : Interface Windows:

L'interface est la partie d'un logiciel qui permet de le manipuler et de prendre connaissance des informations qu'il propose. La caméra est l'objet d'interface entre un monde tridimensionnel et un monde bidimensionnel. Ce n'est néanmoins pas elle qui communique à l'utilisateur l'image qu'elle se fait des astres.

Un autre objet se charge des affichages à l'écran et de l'envoi de messages aux différents objets que j'ai créé. Il s'agit de l'**objet fenêtre**. La fenêtre que j'ai conçue hérite directement de la fenêtre standard offerte par ObjectWindows, dite 'Twindow'.

La fenêtre utilisée dans mon simulateur en est une héritière. En voici la déclaration:

```
TSimulatWindow = object (TWindow)
  ACamera: TCamera;
  Adate: TDate;
  Animation: BOOLEAN;
  Ombres : BOOLEAN;
  Soleil: TBrillant;
  Mercure,
  Venus,
  Terre, Lune, Apollo,
  Mars, Phobos, Deimos: TObscur;

  constructor Init (AParent: PWindowsObject; ATitle: Pchar);
  procedure Paint (PaintDC: HDC; Var PaintInfo: TPaintStruct); virtual;
    procedure WMLButtonDown (var Msg: TMessage); virtual WM_First +
WM_LButtonDown;
    procedure WMRButtonDown (var Msg: TMessage); virtual WM_First +
WM_RButtonDown;
  procedure WMMyTimer   (var Msg: TMessage); virtual WM_First + WM_Timer;

  {Déclarer ici mes procédures associées à chaque item du menu.}
    procedure CMDateInterne (var msg: TMessage); virtual cm_First +
cm_DateInterne;
    procedure CMDateAnimer   (var msg: TMessage); virtual cm_First +
cm_DateAnimer;
    procedure CMDateAvancer (var msg: TMessage); virtual cm_First +
cm_DateAvancer;
    procedure CMDateReculer (var msg: TMessage); virtual cm_First +
cm_DateReculer;
    procedure CMDateQuitter (var msg: TMessage); virtual cm_First +
cm_DateQuitter;

  procedure CMPlanetesRepere (var msg: TMessage); virtual cm_First + 201;
  procedure CMPlanetesNommer (var msg: TMessage); virtual cm_First + 202;
  procedure CMPlanetesOmbres (var msg: TMessage); virtual cm_First + 203;

  procedure CMCameraCabrer (var msg: TMessage); virtual cm_First + 301;
  procedure CMCameraPencher(var msg: TMessage); virtual cm_First + 302;
  procedure CMCameraTourneGauche(var msg: TMessage); virtual cm_First + 303;
  procedure CMCameraTourneDroite(var msg: TMessage); virtual cm_First + 304;
  procedure CMCameraAvant(var msg: TMessage); virtual cm_First + 305;
```

```
  procedure CMCameraArriere(var msg: TMessage); virtual cm_First + 306;
  procedure CMCameraGauche(var msg: TMessage); virtual cm_First + 307;
  procedure CMCameraDroite(var msg: TMessage); virtual cm_First + 308;
  procedure CMCameraHaut(var msg: TMessage); virtual cm_First + 309;
  procedure CMCameraBas(var msg: TMessage); virtual cm_First + 310;
  procedure CMCameraPosition(var msg: TMessage); virtual cm_First + 311;
end;
```

Ma fenêtre, dite TSimulatWindow, est une fenêtre Windows qui a en plus:

Une date,
Une caméra,
Une série d'astres (un brillant, plusieurs obscurs),
Deux indicateurs lui permettant de savoir:
Si les astres obscurs doivent afficher leurs cônes d'ombre
Si la date doit être animée automatiquement.

Chapitre 1 : Construction:

Section 1 : Construction de la fenêtre au sens strict:

Au sens strict, une fenêtre, telle que fournie par Windows, est une zone rectangulaire blanche pouvant être:

déplacée à l'écran,
maximisée de sorte qu'elle occupe tout l'écran,
icônisée dans un coin de l'écran,
recouverte par une autre, partiellement ou totalement,
etc...

Sa construction la rattache à une fenêtre parent dont elle dépend. En l'espèce, mon programme ne gère qu'une seule fenêtre. Le pointeur sur la fenêtre parente est donc mis à **nil**. Le constructeur lui donne également un titre: '**Simulateur Astronomique 3D**'

Certains champs de l'objet fenêtre offert par Windows sont publics. Ils peuvent donc être manipulés de l'extérieur. Il s'agit notamment des champs définissant:

Les coordonnées du coin supérieur gauche,
La largeur et la hauteur.

Si le programmeur ne leur assigne aucune valeur, la fenêtre se place n'importe où selon n'importe quelle largeur et hauteur. Personnellement, je place le coin supérieur gauche en (0, 0)[24] et je lui donne des dimensions de 1024 X 768 pixels.

A ce stade, le simulateur est muni d'une fenêtre blanche sur laquelle rien ne se passe.

Section 2 : Constructions des objets propres au simulateur:

§1 : Astres

1 ° Construction

Lors de son initialisation, ma fenêtre va initialiser chaque astre en les hiérarchisant selon un arbre et en leur donnant leurs caractéristiques propres.

Une seule ligne suffit à créer un astre[25]. Par exemple:

```
Soleil.init     ('Soleil',     30,  255, 255, 000);
```

[24] Contrairement aux usages géométriques, l'origine d'un écran (ou d'une fenêtre) est en haut à gauche et non en bas à gauche. On compte positivement de la gauche vers la droite et du haut vers le bas.
[25] Il faut en réalité deux lignes pour créer un astre: Une dans la partie interface définissant l'astre comme un champ de la fenêtre et une autre dans le constructeur pour l'initialiser.

```
Mercure.init('Mercure',    12,                  060, 120, 255, @Soleil,      060,
88, 5, 0);
Venus.init   ('Vénus',   16,  232, 232, 000, @Soleil, 100, 225, 0, 38);
Terre.init   ('Terre',   16,  000, 255, 255, @Soleil, 200, 365.256361, 00, 38);
Lune.init    ('Lune',    10,  065, 065, 128, @Terre, 040, 27.3216609, 20, 00);
Apollo.init  ('Apollo',  03,  032, 032, 032, @Lune, 015, 10, 00, 00);
Mars.init    ('Mars',    12,  255, 000, 000, @Soleil, 400, 720, 20, 200);
Phobos.init  ('Phobos',  06,  010, 010, 010, @Mars, 015, 30, 5, 0);
Deimos.init  ('Deimos',  10,  000, 255, 127, @Mars, 040, 60, 10, 0);
```

Ces lignes sont extraites du constructeur de ma fenêtre. Construire un astre revient à lui donner un nom, un rayon et une couleur. Il s'agit des trois seuls paramètres requis pour construire un astre brillant[26].

La couleur est elle même composée de trois données. Il s'agit des forces des canons rouge, vert et bleu, les trois couleurs fondamentales. Chaque canon est mémorisé sur un octet. Le blanc est donc composé de 255 dans chaque canon.

Pour les astres obscurs, il faut en outre indiquer:

L'adresse de l'astre central,
Le rayon de l'orbite,
La période de révolution, en jours,
L'inclinaison sur l'écliptique,
Le déphasage, pour ajuster l'astre correctement sur son orbite.

Au terme de cette initialisation, les astres sont initialisés en un arbre dont le soleil est la racine. L'adresse du soleil est donc capitale pour adresser un message à l'ensemble du système solaire car se message sera propagé par des algorithmes récursifs (Cfr. Supra). Il en résulte une grande facilité pour le programmeur qui ne doit pas connaître ici le nombre d'astres du système solaire ni se soucier d'adresser le message à tous les astres.

2 ° Positionnement

Le constructeur de la fenêtre ne se limite pas qu'à hiérarchiser les astres et à leur donner des paramètres orbitaux. Une seconde étape a lieu immédiatement après l'initialisation de la date consistant à envoyer la date fraîchement construite au soleil afin qu'il se positionne (ce qui ne change rien pour lui, puisqu'il est toujours à l'origine), et qu'en tant que racine de l'arbre, il propage l'ordre de positionnement à tous les astres. Car un astre nouvellement construit se place en (0, 0, 0) tant qu'il n'a pas reçu d'ordre de positionnement. Cette situation ne doit pas durer afin que l'utilisateur ne puisse à aucun moment voir un système solaire où tous les astres sont agglutinés à l'origine.

§2 : Date

L'initialisation de la date est considérablement plus simple. Aucun paramètre n'est nécessaire, elle s'initialise spontanément à la valeur contenue dans le setup. (Cfr. Supra, le constructeur de la date.)

§3 : Caméra

1 ° Position

[26]Rappellons ici que le nom donné à l'astre brillant est inutile car le constructeur d'un astre brillant lui donne d'office pour nom 'Soleil' quel que soit le nom donné par paramètres.

La caméra est initialisée de manière telle que le Soleil soit dans son axe de vision. Cela consiste à lui donner une position et une visée telle que le Soleil apparaisse au milieu de l'écran. Personnellement, j'ai choisi de la placer en (-2600, 0, 200) J'ai donc du donner les valeurs 0 à l'azimut et 355 à l'angle de hausse pour qu'elle regarde dans la direction du Soleil. Ultérieurement, la caméra pourra puiser ces valeurs dans un fichier disque pour reprendre la même position que lors de la séance précédente. Ce n'est qu'à défaut de fichier disque qu'elle s'initialisera de la sorte.

2 ° Astre repère

La caméra reçoit l'adresse du soleil pour rester immobile par rapport à cet astre.

Chapitre 2 : Affichages:

Une fenêtre Windows est un rectangle blanc sur lequel il est possible de dessiner. Cependant, elle ignore les graphismes et textes qu'elle contient. C'est donc en tant que rectangle blanc qu'elle va se redessiner. Il va donc falloir lui apprendre à se redessiner avec des astres. L'héritage va jouer ici à plein:

La fenêtre du simulateur est une fenêtre Windows qui sait qu'elle a des astres.

Section 1 : La méthode 'paint'

Chaque fois que la fenêtre estime qu'elle doit se redessiner, elle appelle sa méthode *paint*. l'héritage permet de compléter cette méthode selon les nécessités du programme.

```
procedure TSimulatWindow.Paint;

var
 ChDate : string;
 AffichDate: PChar;
 PageMemoire: HDC;

begin
 inherited paint (PaintDC, PaintInfo);
 ChDate := adate.humaine;
 AffichDate := @ChDate;
 TextOut(PaintDC, (Attr.w - 100), 10, AffichDate + 1, 10);
 ACamera.DimensionnerFenetre (Attr.w, Attr.h);
 ACamera.Projeter(@Soleil, PaintDC, Ombres);
end;
```

§1 : Notions héritées

1 ° Le contexte d'affichage

Techniquement, on ne dessine pas sur une fenêtre, mais sur un contexte d'affichage. Avant l'affichage de tout graphisme ou texte, il faut donc obtenir un contexte d'affichage et l'associer à la fenêtre. Toutes les instructions graphiques désirant s'exécuter sur la fenêtre doivent mentionner le contexte d'affichage qui lui est associé. Une fois le graphisme terminé, le programmeur doit rendre le contexte d'affichage sous peine de plantage, car le Windows ne peut en offrir que 5.

Le contexte d'affichage est une notion voisine de celle de fichier: Il n'est pas possible de lire ni d'écrire directement dans un fichier figurant sur le disque. Le programmeur qui veut écrire sur un fichier doit créer une variable de type fichier, assigner à cette variable une chaîne de caractères contenant un nom de fichier selon la syntaxe MS-DOS et spécifier le nom de cette variable lors de

chaque *Write* ou *WriteLn*: Par exemple: 'C:\DONNEES\LISTE.TXT'. Une fois les écritures terminées, il referme le fichier.

Un contexte d'affichage est donc un peu comme un fichier ouvert.

2 ° Obtention d'un contexte d'affichage

Lorsque Windows appelle lui-même une méthode *paint*, il obtient un contexte d'affichage et le rend après terminaison de la méthode *paint*. La méthode *paint* de Twindow n'affiche rien sur ce contexte d'affichage.

§2 : Notions nouvelles

La fenêtre de mon simulateur est une fenêtre sur laquelle il y a une date et des astres.

1 ° La date

Chaque fois qu'elle se redessine, la fenêtre de mon simulateur affiche la date en haut à droite. Pour cela, elle questionne l'objet date, qui est un de ses champs et donc connu de toutes ses méthodes Elle lui demande une chaîne de caractères comprenant la date au format humain et l'affiche en haut à droite.

La fonction *TextOut* l'affiche sur le contexte d'affichage aux coordonnées déterminées de la manière suivante:

Coordonnée verticale: Une constante.

Coordonnée horizontale: La largeur de la fenêtre moins une constante correspondant sensiblement à la largeur de la date en pixels. Rappelons que cette largeur est *sensiblement constante*[27] car la date s'exprime toujours au format JJ-MM-AAAA comprenant invariablement 10 octets.

2 ° Les astres

La fenêtre va devoir se repeindre avec ses astres.

a) Dimensions de la fenêtre

La fenêtre qui doit se repeindre doit informer la caméra de ses dimensions, car celles-ci peuvent éventuellement avoir changé. La méthode *paint* envoit donc immédiatement les nouvelles dimensions de la fenêtre à la caméra.

b) Positionnement des astres

La fenêtre va ensuite demander à la caméra d'afficher les astres. Elle lui donne pour cela de l'adresse du soleil, le contexte d'affichage et le booléen *ombres* indiquant si les astres doivent afficher leurs cônes d'ombre ou non (Cfr supra: gestion des affichages par la caméra.).

[27] Sous réserve de la fonte de caractères utilisée.

Section 2 : Le message d'affichage

La fenêtre appelle sa méthode *paint* chaque fois qu'elle reçoit le message WM_PAINT. Ce message est envoyé à la fenêtre lorsque son contenu n'est plus d'actualité.

Certains ordres de rafraîchissement peuvent provenir de Windows, d'autres de l'application.

§1 : Ordre de Windows

Windows peut ordonner à une fenêtre de se repeindre pour une des raisons suivantes:

1 ° Redimensionnement de la fenêtre

Une fenêtre est redimensionnée lorsqu'on effectue un tirer-lâcher[28] dans sa bordure ou lorsqu'on la manipule par son item 'Dimension' de son menu système.

2 ° Déplacement

Une fenêtre est déplacée lorsqu'on effectue un tirer-lâcher dans sa barre de titre ou lorsqu'on la manipule par l'item 'Déplacement' de son menu système

3 ° Découverte

Une fenêtre est découverte lorsqu'une fenêtre qui la recouvrait partiellement est rendue inactive et passe donc en arrière-plan. Lorsqu'on clique dans une fenêtre, celle-ci devient automatiquement active et passe à l'avant-plan. Une fenêtre peut encore se découvrir lorsqu'on la sélectionne par le menu de sélection d'applications activé par ALT-TAB ou Control-Esc.

Une fenêtre est également découverte lorsque la fenêtre qui la recouvrait est manipulée de manière telle qu'elle en découvre une partie.

4 ° Maximisation

Une fenêtre est maximisée lorsqu'on clique sur son bouton de maximisation situé en haut à droite ou lorsqu'on active l'item 'Agrandissement' de son menu système.

J'ai remarqué que, lorsqu'on maximise une fenêtre, elle ne met pas à jour ses attributs hauteur et largeur. Il en résulte que ce sont toujours les anciennes valeurs qui sont envoyées à la caméra, qui croit donc erronément pouvoir n'occuper qu'un coin de l'écran. J'attribue ce bug à Windows et non à mon logiciel qui ne fait ici que consulter un champ offert par Windows.

Je n'ai pas traité ce problème, car il est marginal par rapport à mon simulateur. Une solution serait de tester un éventuel drapeau de maximisation et d'aller éventuellement voir dans le système quelles sont les dimensions de l'écran.

[28]Cette opération consiste à cliquer et déplacer la souris en tenant le bouton gauche enfoncé pour le lâcher en un autre point de l'écran.

§2 : Ordre du simulateur

Le simulateur peut aussi décider lui-même de mettre la fenêtre à jour. Il utilise alors la procédure InvalidateRect, munie de l'adresse de la fenêtre à rafraîchir en argument. Cette procédure envoit à la fenêtre le même message WM_PAINT à la fenêtre mentionnée en argument.

J'utilise cette technique plutôt que d'appeler moi-même la méthode paint, car cette solution m'obligerait à acquérir, puis relâcher, un contexte d'affichage.

Le simulateur peut ordonner à sa fenêtre de se repeindre pour une des raisons suivantes:

1 ° Initialisation du simulateur

Lors du lancement de l'application, la fenêtre n'est évidemment pas à jour. Elle doit l'être dès que les astres seront positionnés.

2 ° Modification de la date

Lorsqu'on modifie la date (avance, recul, retour à la date du setup ou à toute autre valeur), il faut immédiatement en avertir tous les astres[29] qui adapteront immédiatement leur position. La fenêtre ne correspond plus à la réalité.

3 ° Modification de la caméra

La caméra peut aussi changer de position ou d'angle de vue. Cette manoeuvre doit immédiatement être perçue sur la fenêtre.

Dans une version ultérieure, la caméra pourra zoomer/dézoomer. Ce qui provoquera également un rafraîchissement de la fenêtre.

§3 : Exemples illustratifs:

La programmation orientée objets permet de délocaliser un problème. Il suffit d'envoyer les bons messages aux bons objets qui réagiront en conséquences. Nous allons analyser la procédure qui réagit à l'ordre de passer au lendemain:

1 ° Passage au lendemain

```
procedure TSimulatWindow.CMDateAvancer;
begin
 Adate.avancer;
 Soleil.positionner (Adate, @Soleil);
 InvalidateRect (HWindow, nil, true);
end;
```

Tout d'abord, la fenêtre dit à la date d'avancer. Ensuite, elle envoit au Soleil l'ordre de se positionner assorti de la date afin qu'il le répercute à tous les autres astres du système solaire. Finalement, la procédure InvalidateRect efface la fenêtre et appelle la méthode Paint qui la redessine avec sa date et ses astres.

2 ° Avance de la caméra.

Pour déplacer la caméra dans une des trois dimensions, il suffit de le lui demander, puis d'ordonner à la fenêtre de se repeindre.

---

[29] C'est à dire uniquement le soleil, qui répercutera récursivement le message.

```
procedure TSimulatWindow.CMCameraAvant;
 begin
  ACamera.Avancer;
  InvalidateRect(HWindow, nil, true);
 end;
```

Chapitre 3 : Réponses aux messages

Ce chapitre décrit comment le simulateur réagit aux commandes de l'utilisateur.

Section 1 : La programmation événementielle

Borland Pascal 7.0 permet de passer de la programmation procédurale à la programmation orientée objets, Windows et son système de messages constitue une programmation événementielle, à la différence de la programmation séquentielle classique.

§1 : Historique: la programmation séquentielle

Antérieurement, un programme était une succession de lignes de codes parcourues séquentiellement par le processeur. Certaines instructions (la boucle, la condition, le goto, ...) pouvaient briser cette séquence et obliger le processeur à exécuter plusieurs fois certaines instructions ou ne pas les exécuter.

Dans ce type de programmation, c'est le programmeur qui prend l'initiative d'un dialogue en affichant ou demandant de l'information au moyen d'instructions ad hoc (WriteLn, ReadLn, ...). L'utilisateur doit faire ce que le programme lui demande au moment où il le lui demande.

§2 : Apport de la programmation événementielle

Ici, au contraire, l'initiative revient à l'utilisateur. L'utilisateur émet des messages auxquels le programmeur répond par une procédure adéquate. Les procédures ne sont plus nécessairement appelées par une instruction d'appel. Elles peuvent être associées à un événement qui provoque leur mise en route. La fenêtre est l'objet qui émet ces messages. Elle incarne l'utilisateur aux yeux du programmeur. Elle comporte des méthodes en réponse à ces messages. Faute de message, le programme est inactif, en attente.

La programmation événementielle permet au programmeur de se focaliser sur l'action à entreprendre lors de la survenance d'un message. Elle le dispense de devoir gérer les attentes et surveiller les différents périphériques.

§3 : Notion d'événement

Un événement est en principe une action de l'utilisateur sur un périphérique. Par exemple, une frappe clavier, un clic souris, un déplacement de la souris , l'introduction d'une disquette dans un lecteur ou un appel entrant sur le modem, ...

Cependant, par souci de décharger le programmeur de tâches de programmations fastidieuses et routinières, d'autres sources d'événements sont apparues. Il s'agit ici d'objets interactifs concrets, purement software, qui seront (presque) considérés comme du matériel. Par exemple, un menu, un bouton, ... Le fait de cliquer sur un menu n'est pas présenté comme un événement provenant de la souris, mais bien du menu lui-même.

D'autres objets peuvent également générer des événements. Par exemple, un *timer*. Il s'agit d'un objet qui émet un message à une cadence régulière. Le simulateur contient un timer qui envoit régulièrement un ordre d'avancement à la date lorsque l'option d'animation est activée.

Nous allons examiner les événements que le simulateur traite et la manière dont il y répond.

Section 2 : Les messages du simulateur

Les messages que le simulateur reçoit sont essentiellement de deux types, selon la typologie abordée dans la section précédente: Les messages de Windows et les messages de la console.

§1 : Les messages de Windows

Il s'agit de messages que Windows est capable d'émettre sans qu'il ne soit nécessaire de créer de ressources au niveau de l'interface à l'aide d'un outil tel que l'éditeur de ressources *Resource Workshop*.

1 ° Paint

Ce message indique que la fenêtre doit être redessinnée. Nous en avons examiné les causes et les conséquences.

2 ° Bouton gauche

Ce message indique que le bouton gauche de la souris est enfoncé. Il contient également les coordonnées de la souris au moment de l'enfoncement. Ces coordonnées sont immédiatement envoyées au Soleil qui les répercutera récursivement à tous les autres astres du système solaire en appelant la méthode *Se Nommer*, définie au niveau du TAstre (Cfr. supra). Tous les astres incluant le clic souris afficheront donc leur nom.

3 ° Bouton droit

Analogue au précédent, ce message indique que le bouton droit de la souris a été enfoncé. Le simulateur y répond en l'envoyant au soleil. A charge pour lui de le répercuter récursivement aux autres, comme le précédent. Toutefois, la récursivité cesse dès qu'un astre répond favorablement à l'appel. Seule l'adresse de cet astre sera prise en considération, c'est à dire envoyée à la caméra qui le prendra pour repère. Cliquer sur le bouton droit est du reste la seule manière de modifier l'astre repère de la caméra.

Si toute l'arborescence a été explorée sans qu'un astre ne réponde favorablement au message, c'est que l'utilisateur n'a pas cliqué sur un astre. Un message d'erreur est alors émis en même temps qu'un son désagréable.

4 ° Timer

Ce message provient à une cadence régulière du timer. La procédure qui le traite teste le booléen *animable*. S'il est vrai, elle appelle la procédure associée au message provenant de la console demandant à la date d'avancer (Cfr. Infra).

Une procédure associée à un événement peut donc néanmoins être appelée de manière tout à fait classique d'un autre endroit du code.

§2 : Les messages de la console

Il s'agit de messages émis par des objets d'interface créés par le programmeur. Ces messages existent donc à l'infini. L'utilisateur dispose de deux outils différents pour les émettre: Le menu et les accélérateurs.

Le menu est une barre de menus déroulants placée en haut de la fenêtre. Il se manipule à la souris.

Les accélérateurs sont une série de combinaisons de touches reconnues par la fenêtre émettant un message identique à certains items du menu. Ils sont provoqués par le clavier sans aucun écho à l'écran.

Chaque item de menu est associé à une méthode de l'objet TSimulatWindow.

1 ° Menu date

a) Interne

Il demande à la date de prendre la valeur du setup de la machine.

b) Animer

Il ne s'adresse pas directement à la date, mais nie le champ booléen *animation* de la fenêtre. Selon la valeur de ce booléen, la méthode associée au timer (cfr. supra) appellera ou non la méthode demandant à la date d'avancer

c) Avancer

Cette méthode demande à la date d'avancer. Elle correspond à l'accélérateur F2.

d) Reculer

Cette méthode demande à la date de reculer. Elle correspond à l'accélérateur SHIFT-F2.

e) Quitter

Cette méthode n'a rien à voir avec la date. Elle provoque la terminaison du simulateur. Elle a été placée en dernière position du menu date pour répondre à un standard d'interface requérant que l'item de terminaison d'un programme soit en dernière position du menu de gauche.

2 ° Menu planètes

a) Repère

Cette méthode demande à la caméra le nom de l'astre qui lui sert de point de repère. Elle l'affiche dans une boîte message.

b) Nommer

Cette méthode envoit au soleil, et donc par récursivité à tous les astres, un ordre inconditionnel d'afficher leur nom. (cfr. supra, Astre.SeNommer). Tous les astres présents à l'écran afficheront leur nom.

c) Ombres

Cette méthode nie le champ booléen *ombres*. Ce champ est envoyé par la caméra aux astres lors de chaque rafraîchissement d'écran pour les informer du souhait de l'interface de voir figurer ou non les ombres à l'écran. Cette méthode correspond à l'accélérateur 'o' minuscule.

3 ° Menu caméra

Ce menu est divisé en trois parties. La première concerne l'orientation de la caméra. La deuxième concerne son déplacement dans l'espace. La troisième concerne l'affichage de son orientation et de sa position à l'écran. Chaque item de ce menu génère le même message qu'un accélérateur. Un utilisateur habitué du clavier n'a donc pas à le dérouler.

a) Orientation

Les quatre premiers items de ce menu concernent l'orientation du regard de la caméra. Ils génèrent les messages permettant de relever, abaisser, tourner à gauche et tourner à droite le regard de la caméra. Les accélérateurs générant des messages identiques sont respectivement:

Shift- flèche haut
Shift- flèche-bas
Shift- flèche-gauche
Shift- flèche-droite

b) Déplacement

Le premier item permet d'avancer le long de l'axe X, Il génère le même message que l'accélérateur 'Flèche haut'.

Le second item permet de reculer le long de l'axe X, Il génère le même message que l'accélérateur 'Flèche bas'.

Le troisième item permet de reculer le long de l'axe Y, Il génère le même message que l'accélérateur 'Flèche gauche'.

Le quatrième item permet d'avancer le long de l'axe Y, Il génère le même message que l'accélérateur 'Flèche droite'.

Le cinquième item permet de monter le long de l'axe Z, Il génère le même message que l'accélérateur 'PgUp'.

Le sixième item permet descendre le long de l'axe Z, Il génère le même message que l'accélérateur 'PgDn'.

c) Position

La caméra peut afficher sa position dans l'espace tridimensionnel, ainsi que la direction (azimut et hausse) de son regard. Cet affichage se fait en-dehors de la méthode 'Paint'. En conséquence, la caméra doit obtenir un contexte d'affichage, y écrire sa position et le rendre. Cet affichage n'a en outre lieu que lorsqu' un message le provoque. Il disparaît donc d'office lors du rafraîchissement de la fenêtre.

Cet item génère le même message que l'accélérateur 'p' minuscule.

Titre 7 : Conclusion

Il est maintenant l'heure d'établir le bilan de ce mémoire, du temps que j'y ai consacré, des objectifs atteints et de ce qui reste à faire.

Chapitre 1 : Objectifs atteints

Section 1 : En programmation

Mon mémoire m'a fait découvrir de nouveaux outils et surtout de nouvelles techniques de programmation. J'ai acquis une bonne maîtrise des objets, tant au niveau de leur conception que de l'utilisation de ceux qui me sont offerts et que je peux compléter par héritage.

L'usage des unités de compilation et de leurs relations *uses* m'est également devenu familier. J'ai créé des unités qui offrent des objets et, très accessoirement, une ou deux fonctions isolées.

Section 2 : En conception

§1 : De logiciels

Située en amont de la programmation, la conception est fortement influencée par l'outil d'implémentation. Mon mémoire m'a également fait progresser dans ce domaine. Il m'a fallu longtemps réfléchir aux objets à créer et à leurs performances. De nombreuses boucles de rétroaction furent nécessaire, en particulier en matière de projection d'un 3D-Point sur un écran. Mon idée initiale était de concevoir un espace tridimensionnel totalement distinct de l'outil de projection en 2D (La caméra).

L'objet 3D-Point ignorerait tout de la manière dont il serait vu. Je me suis rapidement retrouvé dans une impasse car la caméra voyait des astres, mais ne voyait pas les points (ombres, terminateur et éclipses) qui y étaient attachés. J'ai du revoir ma conception des choses et transférer des possibilités de la caméra vers le point. Désormais, l'astre obscur prévient les points qu'il possède de leur nécessité de s'afficher (Cfr. Supra). La conception d'objets est un choix permanent entre d'une part rester aussi proche que possible du monde réel et d'autre part, viser l'efficacité.

§2 : D'interface

Mon mémoire m'a fait découvrir les outils de conception d'interface et la manière d'attacher les interfaces au logiciel auxquelles elles se rapportent. Dans mon domaine, la conception d'une interface n'est pas vraiment prépondérante, vu que tout s'affiche sur une seule fenêtre. Néanmoins, pour l'entrée des manoeuvres de la caméra et de la date, les accélérateurs se révèlent bien utiles. La conception d'interface m'a directement conduit à la programmation événementielle, lorsque j'ai dû attacher des procédures aux messages fournis par l'interface.

Chapitre 2 : Ce qui reste à faire

Un logiciel de cette ampleur est toujours perfectible. Autant dire d'emblée que mon mémoire n'est pas terminé et qu'il ne le sera jamais. Le caractère inachevé d'un projet informatique est d'ailleurs, me semble-t'il, le gage de son succès. Un logiciel n'est plus amélioré lorsqu'il n'intéresse plus personne.

Section 1 : En programmation

Les objets que j'ai créé sont instanciés de manière statique, lors de l'initialisation de ma fenêtre. Dans une version ultérieure, ils seront mémorisés dans une collection enregistrable sur disque. Un outil

d'interface permettra à l'utilisateur de charger un fichier et de modifier l'arborescence des astres et leurs caractéristiques. La seule contrainte sera l'obligation d'avoir un astre brillant en un et un seul exemplaire.

L'utilisateur pourra donc inventer et modifier son système solaire selon sa volonté. Tout sera alloué dynamiquement.

Section 2 : En interface

Les affichages ne se feront plus d'office sur la fenêtre principale, mais sur des fenêtre-filles allouées dynamiquement également. Chaque fenêtre aura sa caméra. La fenêtre principale offrira tous les outils et panneaux de commande de l'ensemble du logiciel.

La caméra sera également modifiée. Au lieu de se déplacer selon les trois axes, elle se déplacera dans son propre repère orthonormé. L'axe de vision constituera l'axe X. Son maniement sera plus proche de celui d'un avion. Il ne s'effectuera plus au clavier, mais à la souris.

Les astres obscurs ne seront plus des sphères lisses, il existe des algorithmes de projection d'une image sur une sphère. Les cartes des différents astres existent. Je pourrais enrichir mon objet Astre Obscur de cette méthode de projection afin que des formes de relief réalistes y apparaissent.

Titre 8 : Remerciements:

Ce mémoire n'est pas que de ma seule conception. Il est d'autres personnes sans qui non seulement il ne serait pas ce qu'il est, mais sans qui je ne serais pas non plus ce que je suis. Puissent-elles être ici remerciées.

Chapitre 1 : A mes parents

Les premières personnes que je remercie sont mes parents. Sans leur amour ni leur attention constante depuis bien avant ma naissance jusqu'à cet instant, il ne m'aurait jamais été possible de surmonter les épreuves. Leur affection m'a donné confiance en moi, ils m'ont permis de faire d'une épreuve un défi et non une angoisse. Grâce à eux, mes échecs ne furent que temporaires. Ils m'ont aidé à les supporter et ensemble, nous nous réjouissons des réussites.

Ce sont eux qui ont éveillé mon intérêt pour une multitude de choses passionnantes dont est fait le monde qui nous entoure. Jamais ils n'ont cessé de répondre à mes innombrables questions.

Mon père m'expliquait tout ce qui est technique et qui fonctionne, comme une voiture, un moteur, une installation électrique, un magnétophone ou tout autre appareil. Ensemble, nous essayions de comprendre. Plus tard, il m'a appris à programmer l'ordinateur qu'il venait d'acheter.

Ma mère me montrait tout ce qui était utile et beau: les animaux, les plantes, la nature et le ciel. C'est elle qui m'a offert mon premier atlas d'astronomie et éveillé mon goût pour les astres.

Il m'est alors spontanément venu l'idée, vers seize ans, de créer un programme qui indiquerait les phases de la Lune. Ce programme était au départ une fonction de la date qui rendait un angle de phase. Cette idée est presque reprise intégralement dans mon mémoire. C'est d'elle que dépend le positionnement des astres. Le langage que mon père m'a appris était le Basic. C'est dans ce langage que j'ai écrit un premier logiciel avant celui-ci. Je l'ai présenté le 03 avril 1996 à l'Institut lors d'une conférence au sujet de l'éclipse de lune qui avait lieu cette nuit-là. Ils y ont assisté ainsi que Monsieur Cherton.

Chapitre 2 : A monsieur Cherton

Le logiciel que j'avais conçu avant d'entrer à l'Institut proposait certains graphiques en fausse 3D. il s'agissait plutôt de perspectives sur des schémas en 2D. Monsieur Cherton m'a proposé de réaliser un logiciel en véritable 3D, il m'a donné les outils mathématiques permettant de gérer un espace virtuel et de le projeter sur un plan de projection bidimensionnel.

En tant que licencié en droit, ma formation mathématique s'est arrêtée avec mes humanités. La programmation en QuickBasic m'a permis d'en conserver certains acquis. Monsieur Cherton m'a donné les outils et principes nécessaires pour mener mon projet à bien.

Il m'a également donné les bases de la programmation orientée objets nécessaires à mon mémoire. J'étais au départ hostile à cette nouvelle manière de programmer, tant je programmais depuis longtemps en programmation Basic procédurale. Qu'il est difficile de changer d'outil quant on en utilise un depuis longtemps!

Il m'a patiemment expliqué cette nouvelle façon de voir les choses. Je la trouve maintenant beaucoup plus efficace car elle me permet de simplifier un problème au départ assez complexe.

Chapitre 3 : A l'Institut d'informatique

Je remercie également les facultés pour la qualité de l'enseignement qu'elles dispensent, pour la disponibilité des professeurs et pour la qualité du matériel qu'elles mettent à notre disposition.

Titre 9 : Bibliographie

Pour rédiger mon mémoire, l'essentiel n'est pas de lire, mais de comprendre. En astronomie, ma première source de renseignements fut d'observer le ciel. C'est de cette manière que j'ai compris les principes exposés dans les ouvrages que je lisais déjà bien avant de venir à l'Institut.

Il en va de même pour la programmation. Les principes exposés dans les livres sont bien vite oubliés s'ils ne sont pas mis en pratique.

L'observation du ciel et la pratique de la programmation sont mes deux références principales. Certains ouvrages m'ont aidé à les développer. J'en possède certains depuis des années et j'y découvre de nouvelles choses à chaque lecture.

Chapitre 1 : Ouvrages astronomiques

- Encyclopaedia Universalis, Atlas d'Astronomie, édition 1993.
- Encyclopaedia Universalis, Atlas de l'Espace, édition 1987.
- Atlas d'Astronomie Larousse, Philippe de la Cotardière, 1981

Chapitre 2 : Ouvrages de programmation.

Section 1 : Antérieurs à mes études à l'Institut

Mode d'emploi de QuickBasic, Jean-Claude DESPOINE, Edition SYBEX, 1991
Aide mémoire de MS-DOS, Edition Marabout, 1992.
Aide-mémoire de QuickBasic, Bernard FRALA, Edition Marabout, 1992.

Section 2 : Dans d'autres langages que le Pascal

Le langage C++, Bjarne STROUSTRUP, Addison Wesley, 1992.

Section 3 : Programmation en Pascal

Programmer en Turbo Pascal 7.0, Claude DELANNOY, Edition Eyrolles, Paris 1995.

Section 4 : Programmation sous Windows

ObjectWindows, Guide du Programmeur, Borland, 1992.