Michel BEIGBEDER
Ecole Nationale Supérieure
des Mines de Saint-Etienne

Castor : un modeleur 3D

RESUME

Castor, à paritr du modèle CSG (Constructive Solid Geometry), accomplit deux fonctions :

• d'une part, il interprète un fichier contenant une description des arbres CSG,

• d'autre part, il facettise ces objets de façon à les transmettre, par l'intermédiaire d'un tube UNIX, à un programme de visualisation de polygones.

ABSTRACT

Castor, using the CSG (Constructive Solid Geometry) model, has two functions :

• first, it interprets a file which contains a description of CSG trees,

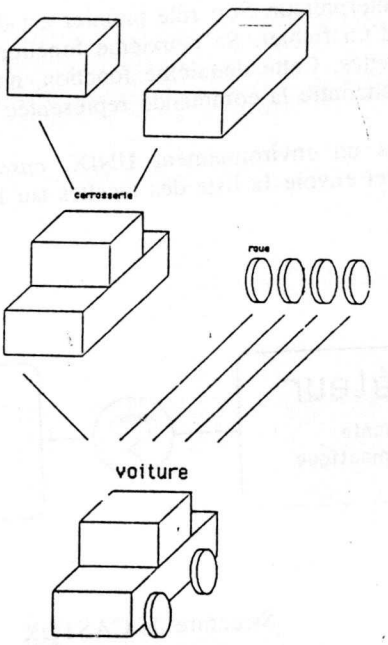
• secondly, it facetizes these objects and transmits them, through a UNIX pipe, to a polygon visualization program.

1. INTRODUCTION

Dans la conception architecturale la production d'images est à la fois un moyen pour vérifier la validité d'un projet, et le résultat à obtenir pour présenter le projet [QUI 84]. L'informatique graphique est tout naturellement amenée à jouer un rôle de plus en plus important dans cette activité de conception [GAN 84]. Toutefois l'architecte doit s'attendre à de nombreuses difficultés lors de la mise en oeuvre d'un outil infographique pour la définition de son projet [GAN 84, QUI 85]. En effet comme dans toute activité informatique nécessitant une quantité importante de données, leur saisie reste un gros problème.

Dans ce qui suit, nous proposons un outil permettant de modéliser et de visualiser d'une façon relativement rapide des objets tridimensionnels simples.

La CAO en architecture est un vaste sujet [QUI 85] et notre outil n'a pas la vocation d'un logiciel de CAO. Il doit simplement permettre d'obtenir des vues sur l'allure générale d'un bâtiment ou d'un projet.



Exemple de modélisation CSG

figure 1

La modélisation CSG (Constructive Solid Geometry) [REQ 80] permet de construire des objets complexes en combinant à l'aide d'opérateurs booléens (union, intersection, différence), des primitives géométriques (cube, cylindre, sphère,...) modifiées ou déplacées par des transformations géométriques (translation, rotation, homothétie,...).

Ce type de description peut être utilisé par plusieurs algorithmes de visualisation avec éliminations des parties cachées :

- z-buffer
- liste de priorité [JAN 83]
- tracé de rayon [JAN 83]
- scan-line [ATH 83]

Castor est un interprète qui utilise ce mode de description géométrique d'objets tridimensionnels.

Dans le paragraphe qui suit, nous allons décrire :

- l'interface entre le système et *castor*,
- la représentation interne des objets,
- la syntaxe de définition.

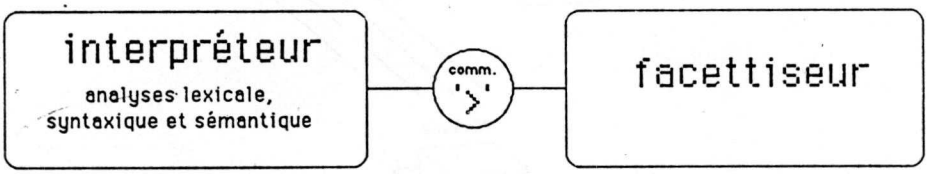
Dans le troisième paragraphe, nous donnerons des exemples et dans le dernier paragraphe, nous proposerons des évolutions envisageables pour ce logiciel.

2. CASTOR

2.1. L'INTERFACE ENTRE LE SYSTEME ET CASTOR

Castor est un interpréteur. Son rôle premier est d'effectuer une analyse lexicale, syntaxique et sémantique d'un fichier. Sa deuxième fonction est de transformer les descriptions d'objets en liste de facettes. Cette deuxième fonction, nommée *facettisation*, est appelée par l'interpréteur lorsqu'il rencontre la commande représentée par le caractère '>'.

Programmé dans un environnement UNIX, *castor* prend en fait l'entrée standard comme fichier d'entrée, et envoie la liste des facettes sur la sortie standard (cf figure 2).



Structure de CASTOR

figure 2

De ce fait il agit comme un *filtre* au sens de UNIX [BOU 83]. Par ailleurs, lors de la détection d'erreurs, des messages sont dirigés vers la sortie standard d'erreur.

Ce mode de fonctionnement permet d'utiliser au mieux les possibilités du système d'exploitation UNIX.

Le fichier d'entrée peut être traité par d'autres filtres avant d'être interprété par *castor*. Le filtre que nous avons employé le plus souvent est le préprocesseur du langage C, à savoir */lib/cpp*. Ce préprocesseur permet de définir et d'utiliser des *macros*.

La sortie, quant à elle, peut être dirigée vers un fichier pour un traitement ultérieur ou vers un autre processus. Typiquement, cette sortie est dirigée vers un programme de visualisation *fvisu* et conjointement des commandes de facettisation sont intégrées dans le flot du fichier d'entrée. De cette façon les programmes de modélisation et de visualisation sont distincts. En particulier, le programme de visualisation peut facilement être adapté au périphérique graphique utilisé. On peut aussi utiliser différents programmes d'affichage suivant le rendu désiré (affichage rapide en *fil de fer*, affichage en faces ombrées, affichage avec textures,...).

En résumé, le dialogue s'effectue grâce à *vi* (l'éditeur pleine page de UNIX) ainsi que grâce aux autres outils UNIX (cf figure 3).

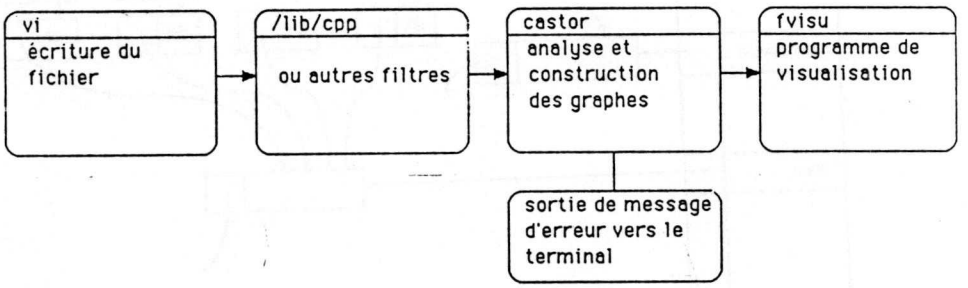


figure 3

L'enchainement des trois programmes */lib/cpp*, *castor* et *fvisu* se fait au moyen d'un shell script.

2.2. LA REPRESENTATION INTERNE

Tout objet porte un nom ; celui-ci est un identificateur alphanumérique commençant par une lettre minuscule. Lors de l'analyse sémantique, une table des identificateurs gérée par *hash-coding* est tenue à jour. Au moyen de cette table, on peut retrouver le pointeur sur le graphe qui définit un objet. Un objet, une fois qu'il a été défini, peut être utilisé pour la définition d'un ou plusieurs autres objets. C'est à cause de cette possibilité que la représentation interne est un graphe orienté et non un arbre.

Par exemple, si les objets *roue* et *carrosserie* ont déjà été définis, nous pouvons écrire :

```
voiture = $U(
    carrosserie,
    @t(X1,Y1,0) roue,
    @t(X2,Y1,0) roue,
    @t(X2,Y2,0) roue,
    @t(X1,Y2,0) roue
    );
```

Ces lignes dans le fichier d'entrée vont définir *voiture* comme la réunion (*$U*) de la *carrosserie* et de quatre *roue*s positionnées grâce aux translations (*@t(...)*) qui utilisent les paramètres numériques *X1*, *Y1*, *X2* et *Y2*.

La représentation interne sera :

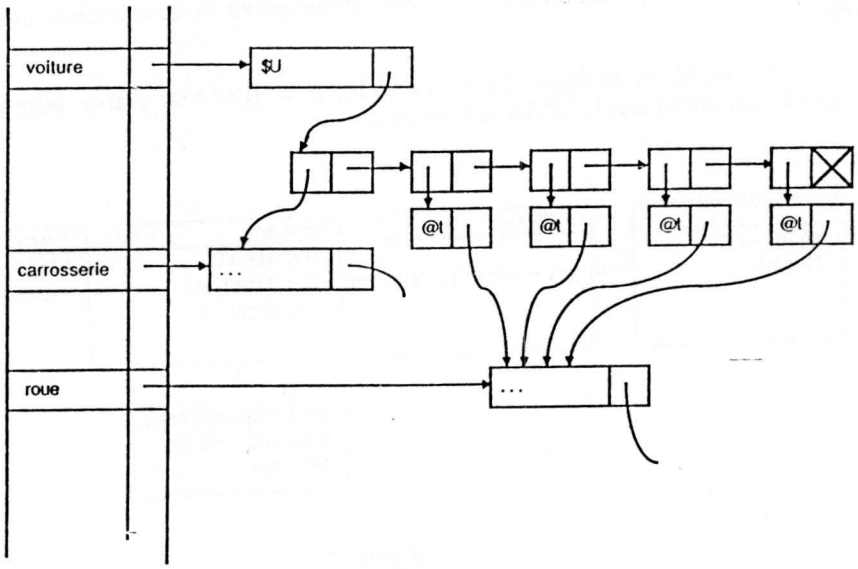


figure 4

Les graphes obtenus sont cependant des graphes sans circuits, et qui de plus possèdent un sommet privilégié : leur racine. Lors de la procédure de facettisation, un parcours en profondeur permet d'atteindre tous les objets élémentaires.

Les noeuds du graphe sont de trois types :

- primitives géométriques,
- transformtations géométriques,
- opérateurs booléens.

Enfin, chaque noeud sait s'il est directement pointé par l'intermédiaire d'un identificateur ; cela permet de documenter pour l'utilisateur le chemin dans le graphe qui aboutit à une primitive géométrique, par exemple :

```
(voiture).U2(roue)
```

désigne dans l'objet *voiture* le deuxième élément de la réunion (*U2*) qui est une *roue*.

2.3. LA SYNTAXE

La syntaxe est très proche de la description interne. Trois caractères préfixent les identificateurs de noeuds :

! introduit un objet élémentaire, par exemple *!cu* va définir un cube, *!sp* une sphère,...
@ introduit une transformation géométrique, *@t* désigne une translation, *@r* une rotation,...
$* introduit un opérateur booléen, *$U indique une réunion, *$I* une intersection,...

La structure brièvement décrite au paragraphe précédent permet de gérer un nombre quelconque de noeuds avant d'arriver aux objets élémentaires, on peut donc utiliser autant de transformations, de combinaisons et d'appels à des objets déjà définis que nécessaire. On peut voir ces possibilités dans les définitions récursives qui suivent :

```
<définition> ::= <identificateur d'objet> = <objet complexe> ;

<objet complexe> ::= <objet élémentaire> |
                     <transformation> <objet complexe> |
                     <union> |
                     <inter> |
                     <diff>

<union> ::= $U ( <objet complexe> { , <objet complexe> }* )
<inter> ::= $I ( <objet complexe> { , <objet complexe> }* )
<diff> ::= $D ( <objet complexe> , <objet complexe> )
```

Signalons, pour terminer, que les objets nécessitent trois paramètres de couleur et que les transformations géométriques ont besoin de paramètres numériques pour être complètement définies. Tous ces paramètres peuvent être soit des nombres réels au format *%lf* du langage C, soit des identificateurs, soit des expressions combinant les éléments précédents. *Castor* possède en effet une table d'identificateurs de paramètres numériques ainsi qu'un évaluateur d'expressions algébriques. Notons que lexicalement un idenficateur de paramètre numérique commence par une lettre majuscule alors qu'un identificateur d'objet commence par une lettre minuscule.

3. RESULTATS

Castor a été utilisé avec succès pour la modélisation de plusieurs projets architecturaux. Nous en présentons ici deux exemples.

Premièrement, nous avons modélisé une maison dans sa totalité. La photo 1 montre la simulation obtenue avec *castor*. La photo 2 prise sur place avec un angle de vue analogue permet d'évaluer l'intérêt de ce type de modélisation.

Le deuxième exemple (cf photo 3) montre un projet d'aménagement d'une voie qui comporte un abribus et un feu de circulation relativement complexes. Le fichier de description a une longueur d'environ 550 lignes et son écriture a nécessité trois journées. L'affichage d'une image comme celle qui est présentée prend environ cinq minutes.

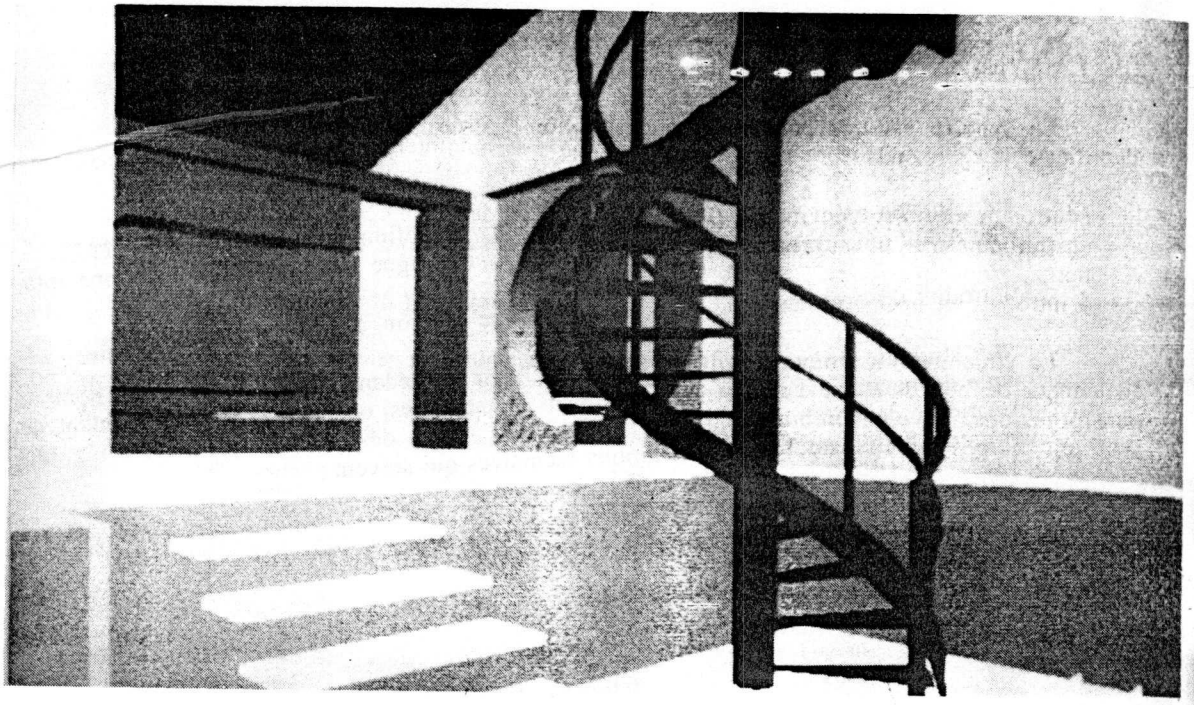


photo 1

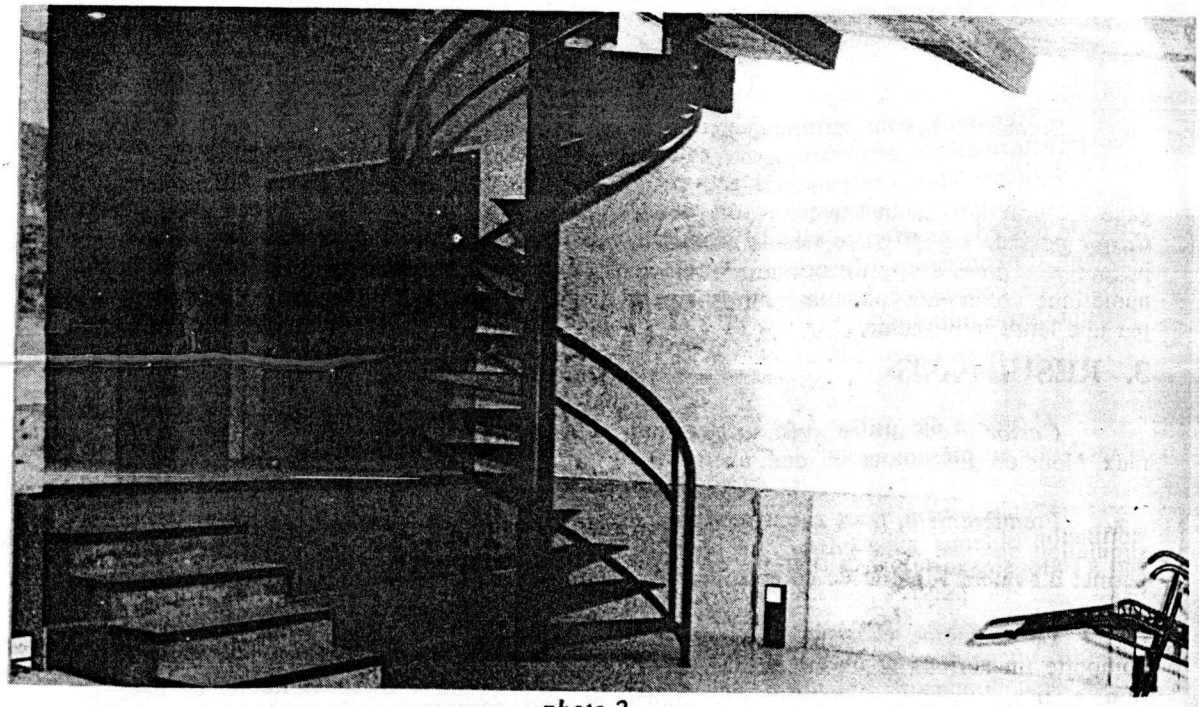


photo 2

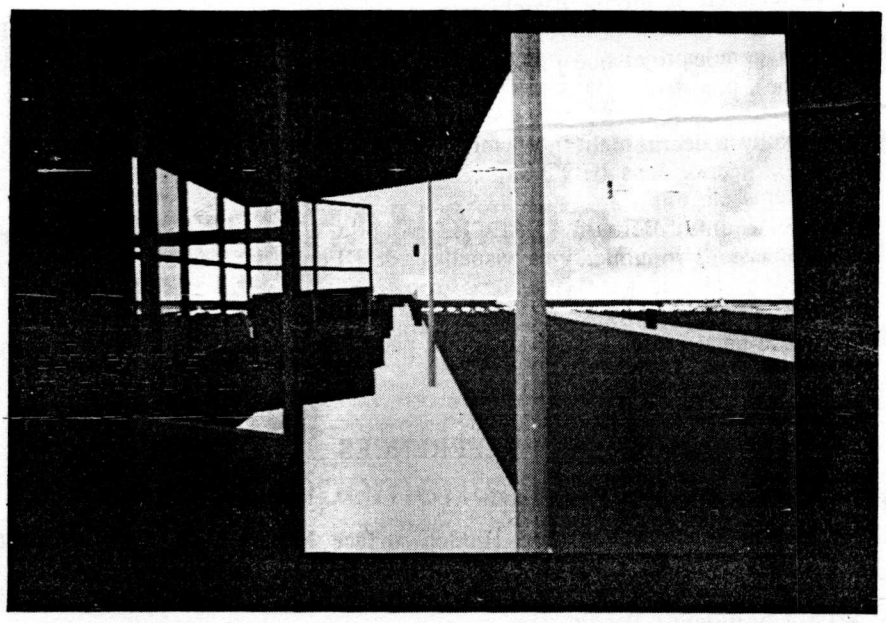


photo 3

4. EXTENSIONS ENVISAGEABLES

Grâce à la structure très modulaire du logiciel, l'ajout d' objets élémentaires au logiciel est aisé. Toutefois, un tel ajout nécessite de fournir une procédure de facettisation avec chaque nouvelle primitive.

Au niveau de la visualisation, il serait intéressant d'adapter le nombre de facettes d'un objet à la surface qu'il occupera sur l'écran. En effet, choisir un grand nombre de facettes pour un objet petit sur l'écran conduit à une perte de temps lors de l'affichage ; par ailleurs, un petit nombre de facettes pour un objet proche met ces facettes trop en évidence. Le problème pour cette extension est qu'elle nécessite que l'on regroupe les programmes de facettisation et de visualisation, programmes qui sont séparés pour le moment et qui communiquent par l'intermédiaire d'un tube UNIX.

Pour pouvoir générer facilement des objets répétitifs comme les escaliers ou encore pour préparer des animations, il serait intéressant de gérer des variables et des structures de contrôle.

Enfin, entrer des données uniquement au moyen d'un fichier nous semble limiter l'utilisation de *castor*. Aussi, envisageons-nous de pouvoir définir les objets d'une manière interactive en positionnant directement les nouveaux objets élémentaires grâce d'une part à une visualisation sur un écran et d'autre part à une tablette à numériser ou une souris. Ceci nécessitera des opérations de recalage des objets car la précision obtenue manuellement ne sera pas suffisante.

REMERCIEMENTS

Je remercie l'atelier d'architecture Franck CHOPIN et Philippe MERLE qui a été le premier utilisateur de ce logiciel en modélisant la maison MERLE que j'ai citée comme premier exemple.

Je remercie le groupe d'architecte et de plasticiens stéphanois LIEU-DIT qui a réalisé le projet d'aménagement de bus sur le CD 518 du Rhône (projet financé par la SEMALY et le TDE du Rhône), projet que j'ai cité comme deuxième exemple.

Sabine Coquillart a réalisé le ciel que l'on peut voir sur la photo 3 grâce à une méthode qu'elle a décrite dans [COQ 85].

Je remercie enfin Bernard PEROCHE pour la lecture critique de cet article, et les membres de l'équipe "communications visuelles" de l'Ecole des Mines de Saint-Etienne pour quelques discussions fructueuses.

Une partie de ce travail a été soutenue par l'Agence de l'Informatique par le contrat numéro 84/486.

REFERENCES

[ATH 83] **P.R. Atherton**, "A Scan-line Hidden surface Removal Procedure for Constructive Solid Geometry", *Computer Graphics*, Vol. 17, No. 3, July 1983, pp. 73-82.

[BOU 83] **S.R. Bourne**, *The UNIX System*, Reading, Addison-Wesley, 1983.

[COQ 85] **S. Coquillart**, "Displaying Random Fields", *Computer Graphics Forum*, 1985, No. 4, North-Holland, pp.11-19.

[GAN 84] **M. Gangnet**, "Projet, image et ordinateur", in *L'image en architecture, les machines à dessiner*, plaquette de l'exposition réalisée par l'Atelier du Patrimoine de la Ville de Marseille et le Gamsau, Laboratoire de recherche de l'Ecole d'Architecture de Marseille Luminy avec le concours du Musée d'Histoire de Marseille et de l'Institut International de Robotique et d'Intelligence Artificielle de Marseille, Novembre 1984.

[JAN 83] **F.W. Jansen**, "A CSG List Priority Hidden surface Algorithm", *Eurographics'85*, Amsterdam, Elsevier Science Publishers B.V., 1985.

[QUI 85] **P. Quintrand, J. Autran, M. Florenzano, J. Zoller**, *CAO en architecture*, Hermès, Paris, 1985.

[REQ 80] **A.A.G. Requicha**, "Representations for Rigids Solids : Theory, Methods, and Systems", *Computing Surveys*, Vol. 12, No. 4, December 1980.

MICHEL BEIGBEDER

Ecole Nationale Supérieure
des Mines de Saint-Etienne

158, cours Fauriel
42023 Saint-Etienne Cédex 2

Tel : 77 42 01 23

Telex : 300 923