PREMIER COLLOQUE IMAGE
Traitement, Synthèse, Technologie et Applications

BIARRITZ – Mai 1984 –

ALGORITHMES GRAPHIQUES PARALLELES
PARALLEL GRAPHIC ALGORITHMS

PELERIN Maxime

L.A. 369 UNIVERSITE DE LILLE I

RESUME

La production d'algorithmes parallèles de synthèse d'images, ceux-ci étant des algorithmes nouveaux ou des versions parallèles d'algorithmes séquentiels, est en étroite relation bilatérale avec l'architecture cible choisie.

Nous présentons dans ce papier des versions parallèles d'algorithmes de tracé de segment et une version nouvelle d'un algorithme de remplissage et de traitement des surfaces visibles.

Pour chaque algorithme; le parallélisme, ainsi que l'influence de l'architecture sur son utilisation, sont évalués.

SUMMARY

The production of parallel algorithms for image synthesis, is in bilateral close touch with the architecture; they are new algorithms or parallel versions of sequential algorithms.

We present parallel versions of line-drawing algorithms and a new version of filling and hidden su-face algorithm.

For every algorithm, the parallelism and the influence of the architecture on its use are evaluated.

ALGORITHMES GRAPHIQUES PARALLELES
PARALLEL GRAPHIC ALGORITHMS

PELERIN Maxime

INTRODUCTION

Les algorithmes parallèles proposés concernent les architectures où chaque élément de l'image est produit par l'ensemble des processeurs élémentaires (architectures MIMD).

Pour choisir un algorithme parallèle pour une machine donnée, il faut prendre en compte deux de ses caractéristiques:

- le parallèlisme de calcul
- le parallèlisme d'accès aux mémoires

Si la première caractéristique est dépendante du nombre de processeurs élémentaires de la machine, la seconde est en étroite relation avec l'organisation des accès à la mémoire des données et à la mémoire d'image(découpage, gestion par un processeur spécialisé...).

Dans la première partie de cette étude nous présentons des versions parallèles d'algorithmes de tracé de segment et montrons le rapport étroit entre un algorithme et une architecture pour leur emploi optimum. Dans la seconde, une nouvelle approche est proposée pour la réalisation du remplissage et du traitement des surfaces visibles.

I-Le tracé de segment de droite

Préliminaires:

Le parallélisme des algorithmes est comparé pour une phase du traitement (calcul et affichage) à l'aide de deux coefficients:

$$C = \frac{\text{nombre de processeurs calculant 1 point}}{\text{nômbre de processeur élémentaires}}$$

$$A = \frac{\text{nombre de processeur affichant 1 point}}{\text{nombre de processeurs élémentaires}}$$

L'idéal étant d'obtenir le parallélisme maximal de calcul et d'affichage, soit respectivement C et A égal à 1 .

Deux approches sont possibles, on peut chercher à privilégier d'abord le parallélisme d'affichage (cas a) et b)) ou au contraire privilégier le parallélisme de calcul (cas c) et d)).

a) Algorithme réparti:

L'utilisation parallèle la plus simple et la moins efficace d'un algorithme, consiste à faire exécuter le même algorithme séquentiel par chaque processeur avec un test permettant de déterminer si le point est dans sa région d'affichage.

exemple:

Si l'architecture de N processeurs a un découpage en sous-ligne, i.e. permettant à chaque processeur d'afficher les points pour une ordonnée y, d'abcisses x= P, N+P, 2N+P,... (P: n° de processeur)

Il faut que chacun des N processeurs teste l'appartenance du point calculé à sa région d'affichage; l'étape de calculs se terminant lorsque la coordonnée x=N ou $y=y\pm1$, on affiche alors la sous-ligne, et ainsi de suite.

Dans cet exemple, le parallélisme d'affichage n'est interessant que si dx est supérieur ou égal à dy (dx,dy: différences descoordonnées extrêmes)

A= si $dx \geqslant dy$ alors dy/dx sinon 1/N

Quelle que soit l'organisation des accés, le parallélisme de calcul sera toujours égal à 1/N, il n'y a pas de parallélisme de calcul car chaque processeur traite tous les points.

b) (n,n) Algorithme (cf. 7):

R. SPROULL utilise une transformation interessante de l'algorithme de Bresenham permettant à chaque processeur P:(0..N-1) de calculer et d'afficher les points d'abcisses: x=P, N+P, 2N+P, ...

Cet algorithme répond à une organisation des accès telle qu'une colonne de la mémoire d'image soit adressée par un seul processeur.

Pour le (n,n) algorithme on obtient:

C=A= si $dx \geqslant dy$ alors 1 sinon dx/dy

Une amélioration possible pour accroître le parallélisme consiste à utiliser le même développement pour y lorsque $dx < dy$.

On obtient alors:

C=1

A= si $dx \geqslant dy$ alors 1 sinon dx/dy

remarque: le coefficient A peut être amené à 1 lorsque $dx < dy$, si l'organisation des accés à la mémoire d'image se fait d'une manière parallèle (ex: processeur spécialisé pour l'affichage).

c) Algorithme dichotomique:

Principe:

Si $A:(x_a,y_a)$ et $B:(x_b,y_b)$ sont les deux extrémités d'un segment S; le point milieu $M:(x_a+x_b/2, y_a+y_b/2)$ appartient à S. Le même raisonnement pour les sous-segments (A,M) et (M,B) nous définit 4 sous-segments de S. En poursuivant le processus, le segment S est entièrement tracé par une méthode récursive et n'utilisant que des opérations d'addition et de décalage.

Primitives:

Pour avoir le même programme pour tous les processeurs élémentaires, des primitives de synchronisation et de communication sont nécessaires.

communication:
- envoi (n°processeur, paramètres): envoi de paramètres d'un processeur à un autre désigné dans la primitive.
- réception (paramètres): réception de paramètres en provenance d'un autre processeur.

synchronisation:
- synchro(n° proc.): envoi d'un signal de synchronisation au processeur désigné.
- attente synchro: une fois cette primitive exécutée, le processeur attent un signal de synchronisation pour reprendre son traitement.

Algorithme:

```
Procédure dicho:(xd,yd,xf,yf:réels)
 début
    tracé des extrémités (xd,yd),(xf,yf);
    pour processeur P=0 à N-1
        I:=1;
        Si P=0 alors
            réception ((xd,yd),(xf,yf));
        Sinon
            attente synchro; /* pour P=1 à N-1 */
            réception ((xd,yd),(xf,yf));
        Fsi
        TQ E(xd)≠E(xf) et E(yd)≠E(yf)
            xm:=(xd+xf)/2;
            ym:=(yd+yf)/2;
            synchro(P+I);
            envoi(P+I,(xm,ym),(xf,yf));
            affichage(E(xm),E(ym));
            xf:=xm;
            yf:=ym;
            I:=2*I;
        FTQ
    fin processeur
fin
```

remarques et notations:

E(x): partie entière de x arrondie

I : permet de lancer un processeur en phase de traitement.

Les calculs peuvent être faits sur 2 registres, l'un contenant la partie entière, l'autre la partie fractionnaire résultant des décalages.

La pente du segment n'intervient pas pour le parallélisme de calcul. Si le nombre de processeurs N est supérieur ou égal à la définition maximale d'une ligne ou d'une colonne de la mémoire d'image, on obtient:

$C=\log_2(\max(dx,dy))/N$ C est inférieur à 1.

C est ici le coefficient de calcul moyen

Une pile commune est nécessaire pour mémoriser N points lorsque $\max(dx,dy) > N$ pour pouvoir poursuivre la récursivité.

Si la sortie sur mémoire d'image est physiquement ou logiquement parallèle (ex: respectivement; réseau d'interconnections, processeur spécialisé), le coefficient d'affichage A est égal à celui de calcul C, sinon, du fait de l'algorithme il sera très voisin de 1/N.

d) Algorithme par analyse différentielle (DDA):

L'algorithme séquentiel de base est le suivant: (on trouvera dans (6) une méthode pour réaliser les calculs en nombres entiers)

```
Procédure DDA (x1,y1,x2,y2:entiers)
var: longueur,i:entiers
     x,y,xincr,yincr:réels
début
    longueur:=ABS(x2-x1);
    Si ABS(y2-y1)>longueur alors
        longueur:=ABS(y2-y1);
    Fsi
    xincr:=(x2-x1)/longueur;
    yincr:=(y2-y1)/longueur;
    x:=x1+0.5; y:=y1+0.5;
    Pour i=1 à longueur
        affichage(E(x),E(y));
        x:=x+xincr;
        y:=y+yincr;
    Fpour
fin
```

La transformation parallèle de cet algorithme nous donne:

```
Procédure DDA // (x1,y1,x2,y2:entiers)
var: longueur, P, N, I:entiers
     x,y,xincr,yincr,dx,dy,pasx,pasy:réels
```

```
début
    affichage(x1,y1);
    longueur:=ABS(x1-x2);
    Si ABS(y1-y2) > longueur alors
        longueur:=ABS(y2-y1);
    Fsi
    xincr:=(x2-x1)/longueur;
    yincr:=(y2-y1)/longueur;
    x:=x1+0.5; y:=y1+0.5;
    pasx:=xincr*N; pasy:=yincr*N;
    Pour processeur P=0 à N-1
          I:=P+1;
          dx:=xincr*I; dy:=yincr*I;
          TQ I ≤ longueur
               x:=x+dx;
               y:=y+dy;
               affichage(E(x),E(y));
               dx:=dx+pasx;
               dy:=dy+pasy;
               I:=I+N;
          FTQ
    fin processeur
fin
```

Dans cet algorithme, tous les processeurs calculent au même moment un point du segment, d'où C=1.

Si les accès à la mémoire d'image sont parallèles A est égal à 1. Sinon l'algorithme nécessite de légères modifications pour avoir un coefficient A différent de 1/N.

e) <u>récapitulatif</u>:

Algorithmes	C	A_{max}
alg.réparti	1/N	SI dy dx dy/dx sinon 1/N
(n,n) avec dév. en y	1	1
dichotomique	$\frac{\log_2(\max(dx,dy))}{N}$ *	C
DDA //	1	1

* N étant à la définition d'une ligne ou d'une colonne de la mémoire d'image

Les meilleurs algorithmes parallèles de tracé de segment sont donc le (n,n) algorithme avec dével oppement en y et le DDA //. La complexité de l'étape de calcul est la même, mais l'implémentation du DDA // est plus simple à réaliser sur des processeurs élémentaires.

II- <u>Remplissage et traitement des surfaces visibles</u>:

a) <u>préliminaires</u>:

Un objet est souvent décomposé en facettes, elles-mêmes exprimées sous forme de polygones.

Un polygone est représenté par une suite de sommets.

ex: $P:(A:(x_a,y_a,z_a),B:(x_b,y_b,z_b),C:(x_c,y_c,z_c))$

Le polygone P est réaliser à l'aide des segments (A,B),(B,C),(C,A).

Pour traiter l'affichage d'un ensemble de polygones il existe 2 techniques principales permettant d'obtenir 2 types de traitement parallèles distincts:

- le premier travaille ligne après ligne; cette méthode ne permet qu'un parallélisme sur le traitement de la ligne (4), ou l'utilisation d'un traitement pipe-line.
- le deuxième élabore dans un premier temps, en parallèle, toute la structure de données utile, pour réaliser le traitement par un parallélisme sur les lignes.

Cette deuxième méthode, malgré un besoin mémoire plus conséquent, permet un parallélisme plus poussé et logiquement plus simple. En effet, si la structure de données est entièrement constituée, chaque processeur peut prendre en charge le traitement d'une ligne, celui-ci étant réalisé de manière séquentielle.

b) <u>Algorithme</u>:

remarque: l'algorithme présenté n'admet pas les polygones qui s'interpénêtrent.

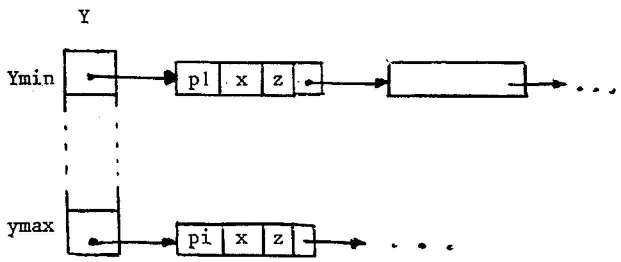
Pour la facilité de compréhension, l'algorithme ci-après, effectue le remplissage avec une couleur uniforme et sans anti-aliasing. Dans ce cas, seul un point d'intersection entre un segments et la ligne est nécessaire, et les coordonnées sont arrondis.

1) <u>La structure de données</u>:

Elle est constituée par une table de pointeurs de lignes et pour chaque ligne d'une liste de quadruplets:

- identificateur de polygone
- coordonnée x
- coordonnée z
- pointeur vers l'elément suivant

Les listes sont triées en x (il n'y a pas de tri en z); le tri en y est réalisé par adressage de la table.



2) constitution de la structure de données:

La constitution de la structure de données est réalisée en parallèle à l'aide d'un algorithme de tracé de segment. Les polygones sont traités un par un, et tous les points des segments sont déterminés en parallèle et placés dans la structure de données en tenant compte des singularités ci-après:

- lorsqu'un segment est commun à plusieurs polygones il n'est pas recalculé, mais ses points sont dupliqués avec des identificateurs de polygone différents
- lorsque l'incrément en y change de signe pour deux segments successifs d'un même polygone, le sommet commun est dupliqué, sinon il n'apparaît qu'une seule fois
- pour des segments horizontaux, les points ne sont pas traités, seuls les sommets sont indiqués et n'apparaissent qu'une fois
- pour les verticaux, tous les points sont traités, et les sommets n'apparaissent qu'une fois
- on détermine l'intervalle (y_{min}, y_{max}) des lignes à traiter, à la constitution de la structure de données

c) Traitement:

Une fois la constitution de la structure de données terminée, le remplissage et le traitement des surfaces visibles pour chaque ligne peut se faire en parallèle.

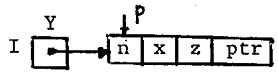
Algorithme général:

```
début
    pour processeur P=0 à N-1
        I:=y_min+P;
        TQ  I≤y_max
            traitement ligne
            I:=I+N;
        FTQ
    fin processeur
fin
```

Le traitement ligne se réalise de manière séquentielle au niveau de chaque processeur élémentaire.

La nouvelle version ci-après, utilise le même principe que l'algorithme " du peintre "; mais elle traite les parties de polygones recouvertes par l'utilisation de la structure de données et non grâce à l'affichage.

traitement ligne:



n: identificateur de polygone
ptr: pointeur
Y: tableau de pointeurs (lignes)
I:indice de ligne
s: pointeur courant de la liste C des polygones recouverts
ref: adresse l'élément de comparaison
p:pointeur de la liste constituant la ligne

```
début
  p:=Y(I);
  Si p≠nil alors
     copie elt(p) dans elt(ref) ;
     p:=p.ptr ;
     TQ p≠nil
        Si p.n=ref.n alors
           /* cas des 2 points du même polygone */
           affichage de ref.x à p.x);
           recherche de z_min dans C;
           Si s≠nil alors
              copie elt(s) dans elt(ref);
              ref.x:=p.x+1;
           sinon
              p:=p.ptr;
              copie elt(p) dans elt(ref);
           fsi
        Sinon
           Si p.x=ref.x alors
              /* un bord commun à 2 polygones */
              Si p.z≤ref.z alors
                insertion en tête(C,p);
              Sinon
                insertion en tête(C,ref);
                copie elt(p) dans elt(ref);
              Fsi
           Sinon
              Si p.z≤ref.z alors
                 /* recherche dans C */
                 s:=tête(C);
                 TQ s≠nil et s.n≠p.n
                    s:=s.ptr;
                 FTQ
                 Si s.n=p.n alors
                     /* polygone recouvert */
```

```
                suppression(C,s);
            Sinon
                insertion en tête(C,p);
                /* début de polygone recouvert */
            Fsi
        Sinon
            /* recherche dans C */
            s:=tête(C);
            TQ s≠nil et s.n≠p.n
                s:=s.ptr;
            FTQ
            Si s.n=p.n alors
                /* mêmes points de début */
                affichage ( de s.x à p.x );
                suppression(C,s);
                ref.x:=p.x;
            Sinon
                /* début de polygone recouvrant */
                affichage(de ref.x à p.x-1);
                ref.x:=p.x;
                insertion en tête(C,ref);
                copie de elt(p) dans elt(ref);
            Fsi
        Fsi
    Fsi
  Fsi
  p:=p.ptr;
 FTQ
FSI
fin
```

Des améliorations sont possibles:

- recherche dans les listes
- utilisation de tableaux au lieu de listes (implique des limitations)
- nouvelle parallélisation au niveau du traitement par ligne

d) Accès à la structure de données:

Si, du fait du traitement spécifique à la synthèse, l'accès à la mémoire d'image ne s'effectue qu'en écriture, d'ou une gestion aisée par un système spécialisé; l'accés aux structures de données doit se réaliser dans les deux sens, d'où un problème de conflits.

La solution actuelle à ce type de problème consiste à utiliser une mémoire partitionée accessible pour chaque processeur à l'aide d'un réseau d'interconnections.

CONCLUSION:

L'emploi d'architectures parallèles en synthèse d'images necessite l'étude d'algorithmes nouveaux ou l'adaptation d'algorithmes séquentiels.

Cette présentation de versions parallèles d'algorithmes de base permet de mettre en évidence deux caractéristiques:

- l'influence réciproque des algorithmes et des architectures parallèles
- une approche possible dans la méthode de choix d'un algorithme parallèle

En effet, l'efficacité d'un algorithme dépend pour une part importante de l'organisation des accès aux différentes mémoires, de même, le choix d'un algorithme parallèle dépend de son parallélisme de calcul et d'affichage (ou rangement en structure de données) et de la complexité de son traitement.

Remerciements:

Je remercie Messieurs V. CORDONNIER, M. MERIAUX et les autres membres de l'équipe d'informatique graphique de l'université de LILLE I pour l'aide et les critiques constructives qu'ils m'ont apportées.

REFERENCES :

(1): G. ALLAIN
Images de synthèse dans les simulateurs de vol.
Bulletin de liaison de l'INRIA, n°88

(2): J. E. BRESENHAM
Algorithm for computer control of a digital plotter
IBM systems journal, vol.4,n°1,1965

(3): F. C. CROW
An approach to real-time scan conversion
AFIPS,vol. 48, 1979

(4): E. FIURME, A. FOURNIER, L. RUDOLPH
A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer
Computer Graphics, vol. 17, n°3, july 1983

(5): M. LUCAS (publié sous sa direction)
La réalisation des logiciels graphiques interactifs
Editions Eyrolles, n°41,1979

(6): W. M. NEWMAN, R. F. SPROLL
Principles of interactive computer graphics
Second edition, Mc Graw-Hill, New York, 1979

(7): R. F. SPROULL
Using program transformations to derive

line-drawing algorithms.
ACM Transactions on Graphics,vol.1,n°4,Oct.1982