PATRICK LARDIN

LE DIAGRAMME VORONOÏ GÉNÉRALISÉ COMME SUPPORT À LA SIMULATION DES ÉCOULEMENTS D'EAU SOUTERRAINE PAR DIFFÉRENCES FINIES INTÉGRÉES

Mémoire
Présenté
À la faculté des études supérieures
De l'Université Laval
Pour l'obtention
Du grade de maître ès sciences (M.Sc.)

Département des sciences géomatiques
FACULTÉ DE FORESTERIE ET DE GÉOMATIQUE
UNIVERSITÉ LAVAL

AOÛT 1999

Résumé

Les outils de modélisation numérique jouent un rôle prépondérant dans la protection de l'environnement. Leur développement, lorsqu'il repose sur les éléments finis (FEM), demeure un exercice difficile, pendant que les approches plus simples, exploitant les différences finies (FDM), sont limitées par des représentations géométriques matricielles. La méthode des différences finies intégrées (IFDM) supporte quant à elle des modèles géométriques élaborés à l'aide de structures irrégulières, tout en appliquant la théorie des différences finies.

Cette recherche vise à appliquer la IFDM au problème de l'écoulement des eaux souterraines en utilisant le diagramme Voronoï comme modèle synthétique. De nouvelles formes discrètes de l'équation d'écoulement prenant avantage des primitives introduites par le diagramme Voronoï généralisé sont développées. Le scénario de Dupuit-Forchheimmer est utilisé pour établir une comparaison entre les résultats donnés par le modèle analytique, la FDM et la IFDM. Ces résultats suggèrent que la technique des IFDM s'appuyant sur les cellules du diagramme Voronoï mérite d'être utilisée comme une alternative aux méthodes courantes.

Avant-Propos

Cette recherche touche à la géomatique, le génie logiciel, l'hydrogéologie, les mathématiques appliquées et la géométrie algorithmique. J'ai assouvi un désir égoïste, celui de réaliser un projet multidisciplinaire. Au contraire de l'effet attendu, ma curiosité ne s'en est trouvée qu'accrue, mon ignorance ayant pris une nouvelle forme, plus grande, fuyant vers l'avant, comme si mon savoir, bien que plus grand, présentait à la fois une frontière beaucoup plus longue...

Au moment où j'ai entrepris ce travail, j'étais loin de me douter du nombre d'heures que j'allais y consacrer, encore moins du nombre exact de disciplines que j'allais devoir maîtriser. Bref, 5 ans plus tard, un mariage plus tard, deux enfants plus tard et un emploi à *Silicon Valley* plus tard, je réalise tout ce que j'ai appris en cours de route. Pas simplement du point de vue académique, mais aussi sur le plan humain. Je me suis découvert des qualités, des défauts, de nouveaux intérêts... j'ai probablement vieilli.

Je tiens à remercier Christopher M. Gold, mon directeur de recherche, pour sa patience, son enthousiasme contagieux et son immense talent de pédagogue. Sans Christopher, mes études graduées n'auraient jamais pris des détours aussi intéressants. Je remercie aussi René Therrien et Alfonso Condal pour leurs remarques éclairées après la lecture du mémoire. Je remercie tout particulièrement Sophie, ma conjointe, et mes enfants Azalée et Étienne pour avoir su me donner du courage aux moments où j'en avais le plus besoin. Et bien évidemment mes parents, pour leur support indéfectible et leurs coups de pieds au c...

TABLE DES MATIÈRES

TABLE DES FIGURES

INDEX DES TABLEAUX

1 Introduction.

1.1 Contexte.

De tout temps, l'histoire de l'humanité s'est vue intimement liée à l'exploitation et la gestion de l'eau. Au cours des deux derniers siècles, avec l'industrialisation et l'apparition des grandes métropoles, l'importance stratégique de l'eau s'est grandement accrue. Aujourd'hui moteur du développement et du maintien des populations, on peut prévoir que la maîtrise des ressources hydriques constituera un enjeu crucial pour les décennies à venir.

Même au Canada, pourtant un des pays les plus riches en eau douce, on observe fréquemment la présence de contaminants dans les lacs et cours d'eau sollicités par des activités récréatives, agricoles ou industrielles. La situation est parfois telle qu'il devient nécessaire d'exploiter les réserves souterraines protégées par d'épaisses couches de dépôts meubles et de roc afin d'assurer un approvisionnement en eau potable à certaines communautés. Au Québec, on estime que 20% de la population utilise de l'eau extraite du sous-sol, bien que ce pourcentage varie d'une région à l'autre en fonction de la disponibilité et de la qualité des eaux de surface. Dans le bassin de la rivière Yamaska, la proportion est de 33% ; dans le bassin de la rivière l'Assomption, elle atteint 65% ; aux Îles-de-la-Madeleine, 100% de l'eau potable consommée est d'origine souterraine (Landry et Mercier, 1992).

Dans l'optique du développement durable, il est admis que l'exploitation des ressources hydriques doit passer par une planification serrée de manière à ne pas compromettre l'aptitude des générations futures à subvenir à leurs besoins. Les réserves souterraines ne font pas exception à la règle puisque contrairement à la croyance populaire, elles ne sont ni stagnantes, ni inépuisables. Elles expriment un fragile équilibre dynamique entre la surface et un milieu poreux, où la percolation des eaux de surface alimente les réserves pendant que l'évaporation, la transpiration végétale, le drainage et le pompage tendent à les vider.

Pour un aquifère donné, les apports et les retraits en eau comptabilisés sur une longue période de temps s'équilibrent, se traduisant par un régime d'écoulement souterrain quasi-permanent. L'élévation à laquelle se trouve l'eau dans le sol - la surface piézométrique - est alors

virtuellement constante, n'étant perturbée que par des fluctuations saisonnières. Il s'agit néanmoins d'un système sensible : lorsqu'une modification est apportée au régime d'écoulement, l'équilibre de la réserve s'en trouve affecté, entraînant à l'occasion des conséquences dramatiques sur l'environnement.

Par exemple, un pompage constant abaisse localement la piézométrie de la formation aquifère. Si le débit pompé demeure inférieur au débit de recharge de la réserve, un nouvel équilibre dynamique est atteint ; la surface piézométrique, bien qu'abaissée, est de nouveau stationnaire (figure 1.1). À l'excès, si le débit pompé est supérieur au débit de recharge du bassin, on assiste à un abaissement incontrôlable de la piézométrie jusqu'à l'épuisement total de la ressource (figure 1.2). Parallèlement à la raréfaction d'une réserve souterraine, on observe généralement la destruction complète du couvert végétal et la migration des populations animales.

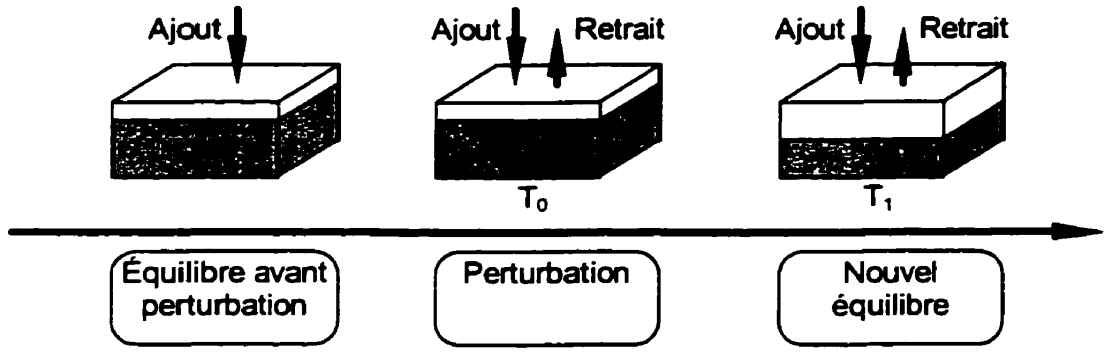

Figure 1-1 Illustration de l'effet sur l'équilibre entraîné par un retrait inférieur à l'ajout.

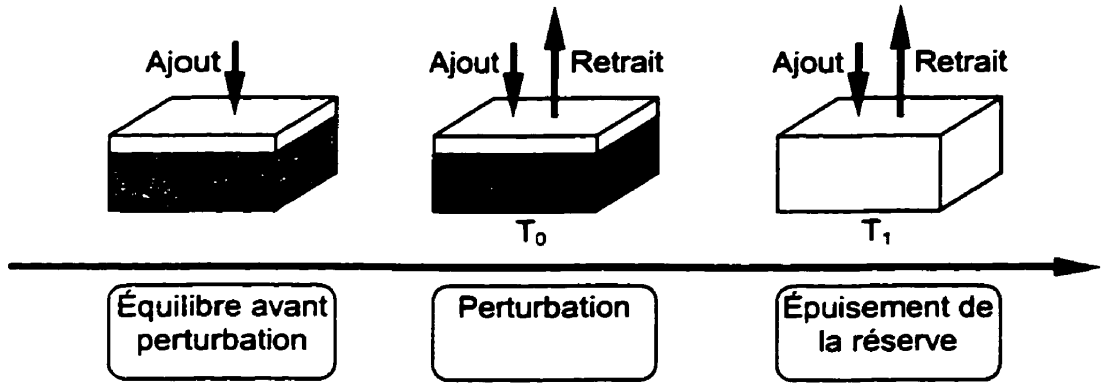

Figure 1-2 Illustration de l'effet sur l'équilibre entraîné par un retrait supérieur à l'ajout.

Inversement, l'irrigation des terres se traduit par un soulèvement de la surface piézométrique. Cette modification au régime d'écoulement, bien que généralement bénéfique aux cultures, peut avoir des effets néfastes sur l'environnement lorsque la montée des eaux souterraines traverse un horizon géologique salin (figure 1.3). En maintes occasions a-t-on observé la salification de terres cultivées et leur transformation en marais salants au détriment des cultivateurs et parfois même d'économies nationales entières (Boonstra et de Ridder, 1981).

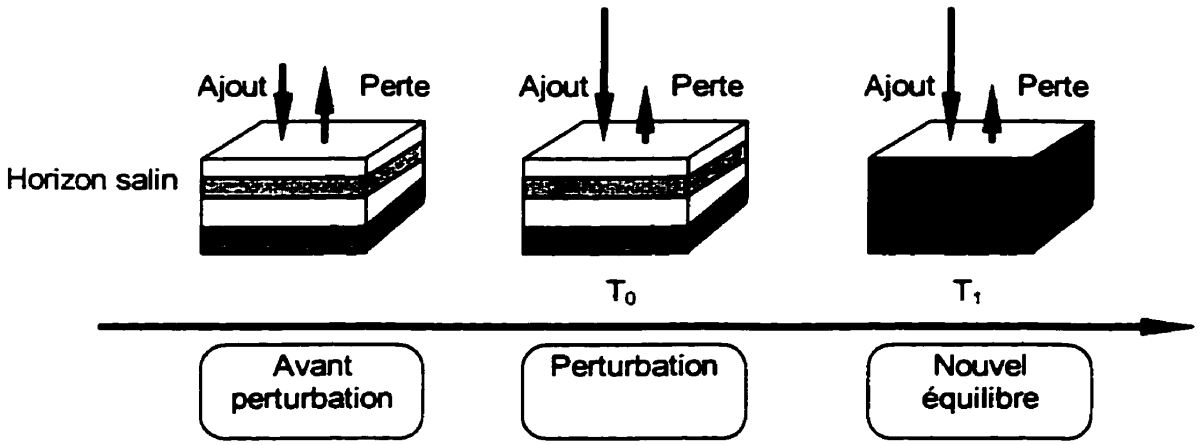

Figure 1-3 Illustration de l'effet d'une remontée saline provoquée par l'irrigation d'un sol minéralisé.

L'infiltration de contaminants chimiques dans les formations aquifères porte aussi atteinte à l'environnement, rendant parfois inutilisables des ressources jadis exploitées pour l'approvisionnement en eau potable de certaines populations. Un déversement accidentel d'essence peut suffire à contaminer, dans un mélange parfait, un volume d'eau égal à un milliard de fois le volume d'essence. L'implantation de dépotoirs en des endroits ou les eaux souterraines effleurent provoque généralement de sévères modifications au régime d'écoulement souterrain, pouvant mettre en contact direct les déchets et l'aquifère.

Comme on le constate, les eaux souterraines forment une réserve facilement affectée par une variété d'activités humaines. Elles sont d'autant plus sensibles qu'elles sont virtuellement invisibles, bien qu'omniprésentes. Les gestionnaires du territoire peuvent facilement omettre de tenir compte de cette réalité environnementale dans leurs efforts de planification si aucun outil n'est disponible pour leur permettre de l'apprécier, de la quantifier et de lui donner un visage.

1.2 Problématique.

Pour être effective, la gestion des eaux souterraines nécessite l'acquisition d'un très grand nombre de données descriptives sur l'état du milieu physique. La complexité liée au stockage, à l'analyse et à la communication de ces données spatiales fait naître de nouveaux besoins en termes de gestion de l'information. En réponse à ces besoins particuliers, un système organisationnel adapté au caractère multidimensionnel de l'information géographique doit être mis en œuvre. Dans ce contexte, les documents de papier - cartes, compilations et autres - conservent certes toutes leurs valeurs pour l'usage courant, mais plusieurs techniques avancées sont maintenant à la disposition des gestionnaires du territoire : modèles numériques, capture de données par télédétection ou encore photogrammétrie, positionnement par satellites (GPS), etc. La démocratisation des technologies de pointes, particulièrement dans le domaine de l'information, joue un rôle déterminant dans la refonte des démarches décisionnelles. À cet égard, il appert que les Systèmes d'Information Géographique (SIG) constituent un facteur central à l'évolution des processus de gestion du territoire.

L'avènement dans un proche avenir de SIG dynamiques distribués, tel que celui mis en chantier par Christopher M. Gold et son équipe de l'Université Laval (Québec, Canada), favorisera le développement d'outils de gestion souples et puissants. Ces architectures fourniront des services de base de données multidimensionnelles à de multiples clients qui auront alors la liberté de mener en parallèle divers types de requêtes spatiales, allant de la navigation assistée aux simulations distribuées (Yang et Gold, 1993).

C'est dans le cadre précis des travaux entrepris par C. Gold que cet effort de recherche s'inscrit, motivé par les aspirations d'usagers désireux d'exploiter le contenu d'une base de données géographiques pour conduire des simulations environnementales de façon directe et transparente. L'érection d'un pont liant les modèles environnementaux aux structures de données spatio-référencées bénéficie de l'éclairage nouveau donné par l'approche «géomatique», comme ce travail tente de le démontrer.

1.3 Projet de recherche.

Les simulations numériques prennent aujourd'hui une place de premier plan dans la gamme des outils utilisés par les gestionnaires du territoire lorsque vient le temps d'étudier les eaux souterraines. La planification des aménagements affectant la ressource - pour son exploitation, son contrôle ou encore sa décontamination - ne saurait se passer de tels leviers d'analyse. Grâce aux solutions numériques données aux modèles mathématiques, il est désormais possible de produire des vues synoptiques de l'hydrosphère qui seraient autrement inaccessibles.

Conceptuellement, une simulation décrit le caractère dynamique d'un système. La précision des «images» qui en sont dérivées est directement liée à la quantité d'information utilisée pour représenter le domaine physique : un modèle défini par très peu d'information résulte en une solution inexacte, alors qu'un second défini par une infinité d'information amène une solution pratiquement exacte. (Anderson et Woessner, 1992).

D'un point de vue fonctionnel, un modèle hydrogéologique passe par la construction d'une représentation géométrique discrète de l'espace physique, structure sur laquelle le modèle mathématique décrivant les écoulements est ensuite «projeté », paramétrisé. La simulation d'un modèle correspond alors à la résolution du système d'équations généré lors du couplage entre le modèle géométrique et le modèle mathématique.

Les solutions numériques les plus populaires en hydrogéologie sont les différences finies (FDM) et les éléments finis (FEM). Le choix d'une méthode ou de l'autre pour solutionner un problème donné dépend principalement des préférences de l'usager (Anderson et Woessner, 1992). D'une part, la FDM représente l'espace en tuiles orthogonales régulières (c.f. figure 1.4). Elle requiert peu d'intrants et le développement de simulateurs fonctionnant sur ce principe est simple (deMarsily, 1986). On peut rapprocher la tessellation de la FDM à un espace rasteur, bien connu des spécialistes de la télédétection et du traitement d'image.

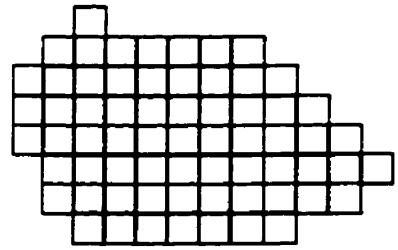

Figure 1-4 Vue en plan de la tessellation de la FDM.

La FEM représente quant à elle le domaine physique à l'aide de tuiles irrégulières généralement triangulaires, tel qu'illustré sur la figure 1.5. Ce modèle facilite la représentation de frontières irrégulières, mais le développement de simulateurs fondés sur ces principes nécessite en contrepartie de fortes aptitudes en calcul et en génie logiciel (Thomas, 1973). Le modèle de l'espace de la FEM se rapproche de l'espace vectoriel exploité par bon nombre de SIG commerciaux.

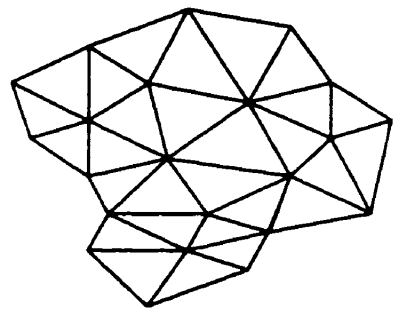

Figure 1-5 Vue en plan d'une tessellation de la FEM.

Les hydrogéologues spécialistes en modélisation numérique tiennent un débat similaire à celui qui tiraille la communauté des SIRS : rasteur ou vecteur ? Pour le folklore, notons que les promoteurs de la maille irrégulière sont en voie de convaincre les usagers en mettant en évidence la liberté qu'ils ont d'utiliser de grandes tuiles là où peu de variations spatiales sont attendues dans l'état du système, et de petites tuiles aux endroits où de forts gradients risquent d'apparaître - tels au voisinage des pompes, des cours d'eau, des lacs, etc. Cet avantage se traduit pour la FEM en économies de mémoire et de temps de calcul par rapport à un modèle de même précision construit pour la FDM.

Quelque part entre ces extrêmes que sont FDM et FEM existe une approche hybride qui constitue un compromis intéressant entre la simplicité de la FDM et la flexibilité de la FEM.

Proposée très tôt pour la modélisation des transferts de chaleur par conduction (MacNeal, 1953), elle utilise un modèle géométrique constitué de tuiles irrégulières (c.f. figure 1.6) et un modèle mathématique simple qui met à profit les calculs par différences finies pour évaluer les débits entre tuiles voisines. Cette technique, baptisée la méthode des différences finies intégrées (IFDM), est considérée par certains comme une généralisation de la FDM pour des tessellations irrégulières (deMarsily, 1986) alors que d'autre l'interprètent plutôt comme un cas de sous-espèce de la FEM (Huyakorn, 1983). La figure qui suit illustre un modèle géométrique typique de la IFDM.

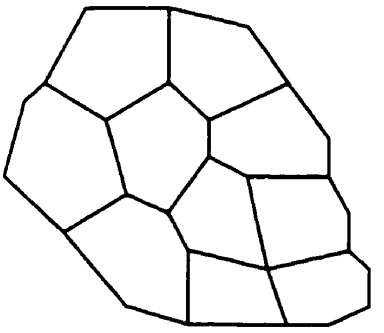

Figure 1-6 Vue en plan d'une tessellation de la IFDM.

L'objectif central de cette recherche est d'explorer le potentiel du diagramme Voronoï généralisé dans le plan comme double support de l'information spatiale et de simulations numériques pour la modélisation environnementale. La problématique des eaux souterraines illustre cette démarche, bien que le cadre ici développé puisse probablement être transposé à d'autres types de simulation reposant sur une subdivision irrégulière de l'espace.

Nous généralisons la IFDM de façon à permettre la conduite de simulations hydrogéologiques à même la structure de diagrammes Voronoï de points, de lignes et de polygones. L'utilisation d'entités linéaires ouvertes et fermées comme nœuds d'un maillage destiné à supporter une simulation numérique constitue une nouveauté dont il n'est nulle part fait mention dans la littérature courante. Cette re-définition de la IFDM s'inscrit dans le prolongement d'un des axes directeurs tracés dans *'Application Challenges to Computational Geometry'*, document publié en 1996 par le *'Computational Geometry Impact Task Force'* :

> « ... *Meshes are used primarily to integrate partial differential equations over complicated domains. The base of numerical solution technologies is surprisingly narrow: finite-element methods, boundary-element methods, finite-difference methods, and Monte-Carlo methods. Are there alternatives to the methods listed above? Recent research suggests that some might lie in finding more clever representations of the underlying geometry which exploit the physics of the problem (in some cases, medial axes and Voronoi methods may come into play) and then mapping the physics onto the new representations of problem geometry. This results into a reformulation of the problem that is both geometric and numeric, as opposed to the purely numeric approaches currently in favor.* »

Au sens large, ce mémoire tente de démontrer que le modèle de données spatiales Voronoï développé selon l'approche géomatique par Christopher M. Gold et son équipe de l'Université Laval (Québec, Canada) présente tous les éléments nécessaires à la réalisation de simulations numériques complexes dans le cadre de problématiques environnementales préoccupantes.

1.4 Présentation du document.

La suite du document se présente comme suit :

- Le chapitre 2 décrit brièvement les efforts faits à ce jour dans le but d'incorporer les simulations numériques dans les SIG.
- Le chapitre 3 présente les grandes familles de modèles en hydrogéologie.
- Le chapitre 4 donne un portrait de la méthode des différences finies intégrées (IFDM) dans une perspective à la fois historique et mathématique.
- Le chapitre 5 introduit le concept de diagramme Voronoï généralisé dans le plan.
- Le chapitre 6 généralise le modèle de la IFDM pour les entités linéaires et polygonales du diagramme Voronoï généralisé dans le plan.

- Une validation de la IFDM pour le scénario de Dupuit-Forchheimmer est donnée au chapitre 7. Les résultats sont comparés à ceux obtenus par la FDM de même qu'à ceux retournés par l'équation analytique de Dupuit-Forchheimmer.

- Les conclusions sont finalement tirées dans le chapitre 8, suggérant au surplus les avenues que pourraient prendre des travaux futurs.

2 État de l'intégration des modèles numériques dans les SIG.

La mise en œuvre de vastes systèmes d'information dans le domaine de l'aménagement des ressources hydriques tire profit de la technologie des SIG depuis une quinzaine d'années déjà. L'utilisation des SIG dans ce contexte va de l'analyse des bassins versants (Sasowsky et Gardner, 1991), à l'étude du cheminement des contaminants agricoles (Baker et Panciera, 1990) en passant par l'identification de zones de protection des puits d'alimentation en eau potable (Rifai et al., 1993).

De telles applications environnementales nécessitent cependant l'utilisation de modèles numériques puissants, parallèlement aux SIG eux-mêmes. À ce jour, on observe que la constitution d'environnements intégrant les simulations numériques aux SIG tend à suivre un nombre restreint d'avenues selon le niveau d'intégration visé entre le simulateur et le SIG. La présente section catégorise les démarches entreprises en fonction du niveau d'intégration SIG - simulateur.

2.1 Intégration faible.

On distingue en tout premier lieu les environnements où un SIG est exploité en tant que pré-processeur et post-processeur, étant principalement responsable de la saisie et du conditionnement des données, de même que de la mise en forme des résultats. Le module de simulation est dans tous les cas un logiciel dont l'exécution est indépendante de celle du SIG, utilisant ses propres structures de données et son propre espace mémoire.

Le synchronisme entre les activités du SIG et du simulateur est assuré par un mode de communication séquentiel relativement fragile : les informations sont transitées par le biais d'un module de conversion bidirectionnel qui transforme les fichiers de données du premier en fichiers de données compatibles au second. Il s'agit d'un niveau d'intégration faible car les deux applications ne peuvent ni communiquer directement, ni partager dynamiquement les données de l'espace mémoire, faisant plutôt usage de la mémoire de masse du système, comme l'illustrent les figures 2.1 et 2.2.

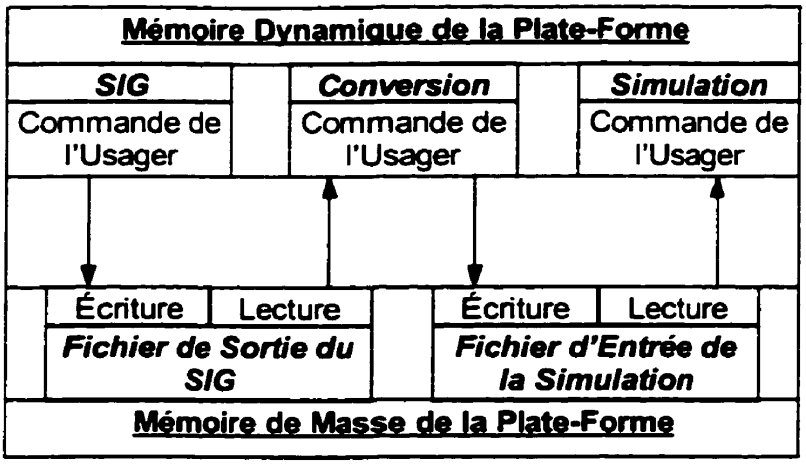

Figure 2-1 Illustration du flux de données entre le SIG comme pré-processeur, le module de conversion et le programme de simulation.

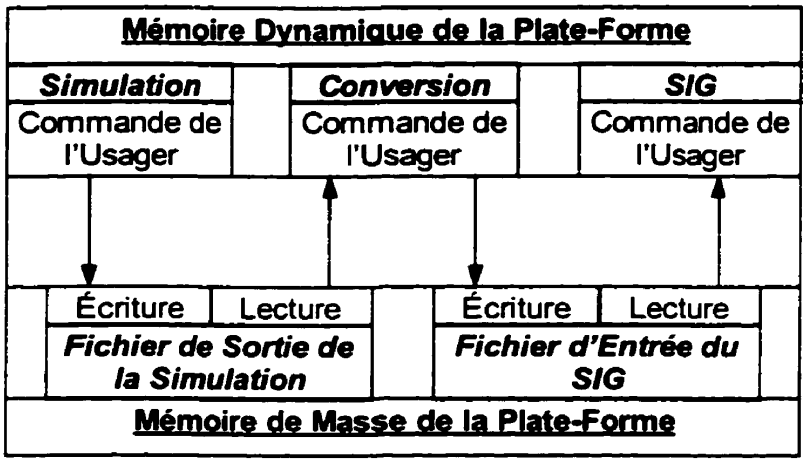

Figure 2-2 Illustration du flux de données entre le programme de simulation, le module de conversion et le SIG comme post-processeur.

Évoluer dans un environnement faiblement intégré implique l'utilisation de plusieurs applications :

- le SIG ;
- le programme de conversion bidirectionnel ;
- le programme de simulation.

Comme une simulation est rarement réussie lors de la première tentative, le processus itératif d'essais et d'erreurs amène l'usager à passer continuellement d'une application à l'autre, à appliquer le filtre de conversion, à retoucher la topologie de la base de données du SIG pour chaque modification apportée à la géométrie du domaine etc. Il s'agit néanmoins de la

méthode d'intégration la plus populaire, phénomène qui s'explique en partie par la disponibilité d'un grand nombre de filtres de conversion dans le domaine public.

Plusieurs travaux publiés portent par ailleurs sur les modifications à apporter à certains simulateurs hydrogéologiques de façon à leur ajouter un filtre de conversion bidirectionnel (Orzol et McGrath, 1993). Les plus récents environnements de simulation, tel le «*Groundwater Modeling System*» (GMS) développé par le département américain de la défense (DoD), sont quant à eux livrés avec les outils de conversion nécessaires pour la lecture et l'écriture de fichiers dans les formats commerciaux les plus populaires : Arc/Info, MicroStation, AutoCad, etc. Il s'agit cependant toujours d'architectures faiblement intégrées dont le flux d'exécution est caractérisé par la réalisation implicite d'une conversion pour chaque entrée/sortie liée au simulateur. La figure 2.3, en opposition à la figure précédente, fait ressortir les différences de cette variante de l'intégration faible dans un scénario où le SIG est utilisé comme post-processeur.

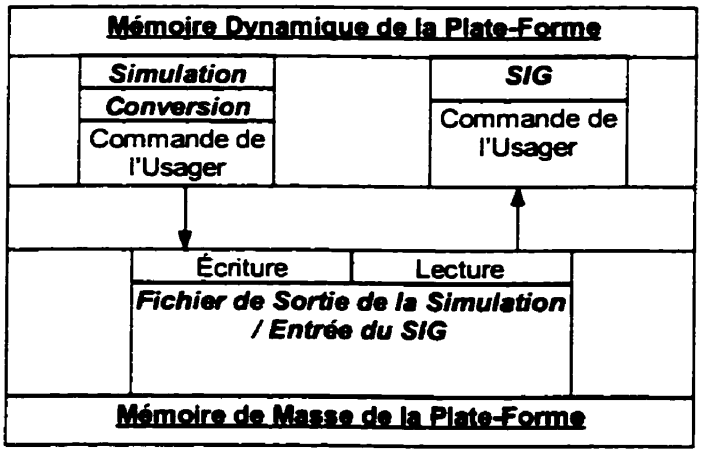

Figure 2-3 Illustration de la dynamique entre les composantes d'une architecture faiblement intégrée où le filtre de conversion est implicite au programme de simulation.

2.2 Intégration forte.

Pour simplifier la tâche des usagers, certains préconisent l'utilisation des langages interprétés (scripts génériques propres aux SIG commerciaux, VisualBasic ou autre) de manière à intégrer de façon tout à fait transparente le simulateur à l'interface-usager des SIG pour ainsi cacher à l'usager la coupure réelle qui existe entre le SIG et le simulateur (El-Kadi et al., 1994).

Cette approche de type «ComponentWare », récemment développée en génie logiciel, fait du simulateur une composante architecturale subordonnée au SIG. Ce niveau d'intégration permet à l'usager de travailler dans un environnement unique, le SIG, où apparaît de nouveaux contrôles à la barre des menus. Un exemple de cette barre des menus nous est donné à la figure 2.4.

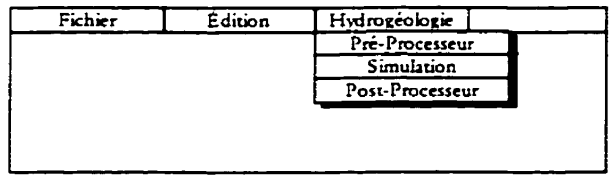

Figure 2-4 Aperçu de l'interface-usager d'un SIG couplé à un simulateur hydrogéologique.

L'intégration forte exploite la modularité des SIG commerciaux et la flexibilité des procédures interprétées. L'usager peut facilement visualiser l'information de son modèle sous une forme graphique, appliquer les outils d'analyse spatiale du SIG aussi bien aux intrants qu'aux extrants, habiller les résultats d'une simulation etc.

Dans un scénario d'utilisation typique, le SIG génère un modèle géométrique complet avant chaque tentative de simulation : une tessellation, généralement orthogonale et régulière, de même qu'une table d'attributs sont construites et paramétrisées par un rééchantillonnage des données descriptives du SIG. Ce rééchantillonnage des paramètres de la base de données géographique pour les besoins de la simulation consomme évidemment temps et ressources, et introduit inévitablement une distorsion de l'information, situation illustrée sur la figure 2.5.

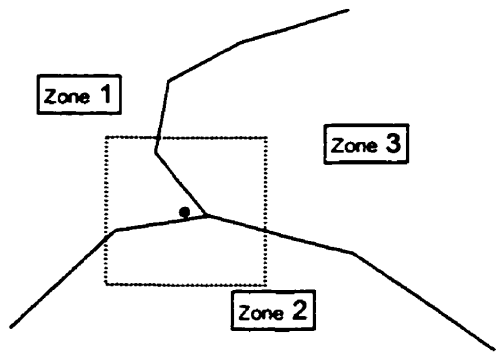

Figure 2-5 Stratégies de rééchantillonnage. La tuile en pointillés de la tessellation cible prendra une valeur en fonction 1) de la plus grande surface recouverte, 2) la position du centroïde, 3) interpolation par moyenne pondérée.

Une fois le modèle constitué, la tessellation et les informations qu'elle contient sont converties et transmises au simulateur par le biais de routines de conversion et d'un protocole de communication basé sur un bus logiciel supporté généralement par le système d'exploitation de la plate-forme. Le résultat de la simulation est finalement retourné au SIG par le même bus logiciel, après une nouvelle conversion. Ces données peuvent finalement être analysées et habillées grâce aux fonctionnalités du SIG mises en ligne.

Il s'agit d'un environnement présentant un niveau d'intégration fort du point de vue architectural. En aucun moment la mémoire de masse n'est sollicitée explicitement, faisant plutôt usage d'un bus logiciel supporté par le système d'exploitation, tel le «*Common Object Model*» (COM) de Microsoft, pour établir la communication entre le SIG et le simulateur. L'usager ne connaît rien des procédures de conversion, des appels au module de simulation, du protocole de communication, etc. La coupure existant entre le SIG et la simulation lui est invisible comme l'indique la figure 2.6.

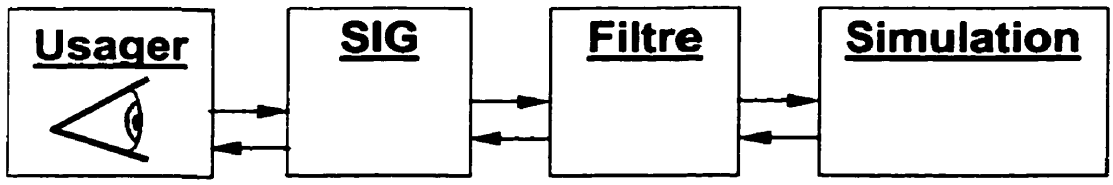

Figure 2-6 Schématisation de la dynamique entre les entités liées dans une architecture logicielle forte. Les flèches symbolisent des transferts de données.

L'intégration forte représente un stade évolutif plus raffiné que l'intégration faible. Néanmoins, le fait que chaque tentative d'une simulation doive passer par la construction d'un modèle géométrique et sa paramétrisation par un rééchantillonnage des données spatiales laisse paraître une faiblesse globale de l'approche. En effet, bien que le SIG soit exploité pour la construction, l'édition et la paramétrisation du modèle, de même que pour l'analyse et l'habillage des résultats, le lien directionnel entre le SIG et le simulateur implique invariablement la construction entière du modèle hydrogéologique, sa conversion et sa transmission, ce qui dénature totalement le sens de l'expression «environnement intégré interactif».

2.3 Intégration complète.

De nouvelles initiatives visant une plus grande fluidité dans l'utilisation et l'interopérabilité des sous-systèmes du SIG ont récemment vu le jour. Ce mouvement a été amorcé par l'avènement de méthodologies architecturales entièrement orientées objet qui ont mené au développement d'outils spécialisés, dérivés des SIG conventionnels et parfaitement adaptés aux besoins particuliers des développeurs de logiciels. De tels outils permettent la mise en œuvre de solutions aussi pointues que celle de la simulation environnementale.

Ces architectures ouvertes permettent la libre extension du modèle objet à travers l'utilisation d'interfaces fonctionnelles standards. La connectivité entre les différentes parties du système est potentiellement assurée par un bus logiciel - élevant en abstraction la charge associée au système d'exploitation – ce qui favorise le développement d'architectures parallèles, voire totalement distribuées.

L'extensibilité du modèle objet permettent au simulateur de «voir» et de manipuler les sous-systèmes du SIG et ses entités dans une architecture complètement intégrée, de générer rapidement et sans conversion des modèles hydrogéologiques consistants et des simulations complexes directement à partir des données descriptives de la base de données (Deckers, 1995). L'efficacité de cette approche contribue à raccourcir le laps de temps séparant deux tentatives de simulation, ce qui est tout à fait souhaitable dans le contexte d'un environnement interactif.

Le Système d'Information Hydrogéologique de l'Institut Hollandais des Géosciences Appliquées constitue à ce chapitre un exemple remarquable, couvrant la totalité du territoire hollandais et mettant à la disposition des scientifiques de l'Institut la possibilité de simuler pratiquement en temps réel l'état des ressources hydrique du pays tout entier à travers le SIG orienté-objet Smallworld (Deckers, 1995).

3 La modélisation en hydrogéologie.

Dans la présente section, nous présenterons les principales avenues empruntées par les hydrogéologues en réponse à leurs besoins en outils de simulation. Les modèles seront présentés en ordre croissant de complexité dans une séquence logique, et le lien qui existe entre chacun d'eux sera mis en évidence. Cette révision sommaire vise à introduire la IFDM d'une façon pratique, prenant soin au passage de réviser les bases physiques et mathématiques requises à la compréhension de la suite du document.

3.1 Le modèle unicellulaire.

Le modèle hydrogéologique le plus simple qui puisse être construit est constitué d'un volume de matériau poreux de dimension finie et de composition homogène. Cette «cellule» est caractérisée par un état dynamique initial et un état statique intrinsèque. L'approche du modèle unicellulaire consiste simplement à appliquer les lois de la thermodynamique classique sur un système ouvert pour suivre l'évolution de son état dans le temps (Bear, 1979).

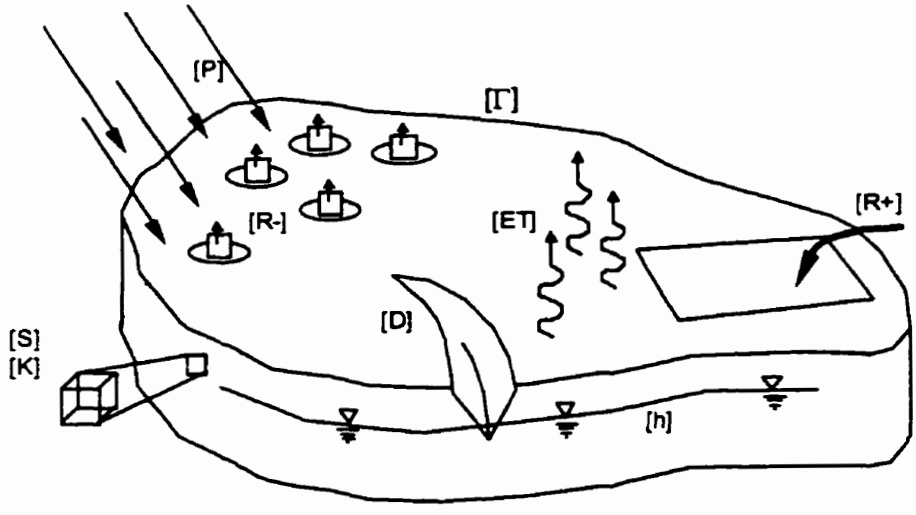

Figure 3-1 Illustration d'un modèle unicellulaire simple (d'après Bear, 1979).

Considérons le modèle illustré à la figure 3.1, soit un bassin délimité par une ligne de crêtes topographiques [Γ]. On observe que les précipitations qui s'y abattent [P] sont évacuées sous deux formes distinctes : état de vapeur et état liquide. La première fraction [ET] est libérée vers l'atmosphère par évaporation ou par transpiration végétale. L'autre partie, la fraction

liquide, s'infiltre dans le sol et se dirige vers la rivière qui constitue l'unique exutoire naturel [D] du bassin. Remarquons que l'eau de pluie qui tombe au-delà de la ligne de crête [Γ] ne participe ni par ruissellement ni par infiltration au régime d'écoulement de la cellule. Pour cette raison, on considère que les lignes de crête agissent à la manière d'une frontière imperméable.

Quelques installations sollicitent la réserve souterraine. En certains endroits des puits d'alimentation en eau potable exercent une pression négative sur la réserve [R-], pendant qu'ailleurs de l'eau est importée pour l'irrigation des cultures, ce qui induit une pression positive [R+]. Le niveau piézométrique moyen [h] reflète l'état d'équilibre dynamique du système ; son élévation correspond plus ou moins à celle de la rivière, l'eau souterraine étant partout en contact hydraulique avec les eaux de surface.

Supposons un instant que le régime d'écoulement soit modifié par l'ajout d'une station de pompage. Logiquement, nous pouvons prévoir que cet ouvrage aura pour effet de diminuer le débit de la rivière [D] et d'abaisser le niveau piézométrique moyen [h]. Avant de pouvoir quantifier le phénomène, il est important de comprendre que le sol, constitué d'une fraction solide et d'une fraction poreuse, joue le rôle d'une éponge dont la capacité à retenir l'eau et la laisser circuler est modulée par le rapport volumique de sa fraction poreuse [S].

La relation entre les composantes du modèle unicellulaire est simple. Au regard de la figure 3.1 et du scénario développé plus haut, il apparaît que l'équation mathématique gouvernant le système résulte du bilan des entrées et des sorties d'eau, mis en relation avec la quantité d'eau qui s'est ajoutée ou a été soutirée aux réserves du bassin. Soit (d'après Bear, 1979) :

$$\text{Entrées - Sorties = Masse accumulée (ou extraite)}$$

$$\left\{P+\sum[R+]\right\}-\left\{ET+D+\sum[R-]\right\}=S\frac{\Delta h}{\Delta t}$$

Éq. 1

où les entrées sont constituées des précipitations et des ajouts par irrigation ou injection, et les sorties de l'évapotranspiration, du débit sortant de la rivière et des stations de pompage. Le

taux de variation de la piézométrie dans le temps est quant à lui modulé par la capacité d'emmagasinement du bassin qui reflète la capacité du milieu poreux saturé à libérer de l'eau sous l'action de la gravité [S].

Dans la pratique de l'hydrogéologue, une simulation menée à l'aide du modèle unicellulaire peut suffire lorsque seul un ordre de grandeur est recherché ou encore une solution grossièrement approchée. La relation topologique d'inclusion doit être préservée lors de la construction d'un modèle de cette nature, sans aucun argument géométrique. Les faiblesses de l'approche unicellulaire sont évidentes lorsque l'on cherche une solution précise concernant la répartition de l'eau, son mouvement ou le cheminement de particules. Pour trouver réponse à ces questions, des outils de modélisation plus subtils sont nécessaires.

3.2 Le modèle analytique

Le modèle analytique comble certaines des lacunes du modèle unicellulaire en donnant une image précise de la piézométrie en tout point du domaine. Outre les entrées, les sorties et le coefficient d'emmagasinement, le modèle analytique tient compte du principal paramètre de contrôle de la distribution de l'eau à l'intérieur du système, soit la conductivité hydraulique [K] entre les pores du sol. Le modèle analytique pave la voix aux solutions relatives au mouvement de l'eau ou encore à celles ciblant plutôt le cheminement des particules.

En vérité, il n'existe pas de modèle analytique universel, mais bien plusieurs modèles tous construits selon les mêmes principes : la combinaison de l'équation de conservation de la masse à celle de Darcy, loi empirique énoncée en 1856 par le Français Henry Darcy alors qu'il étudiait les fontaines de Dijon.

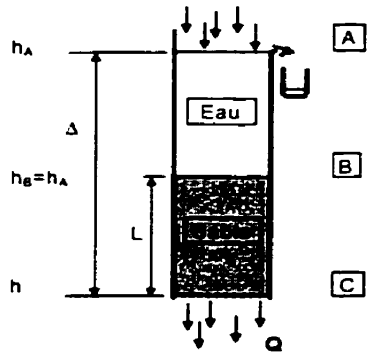

Figure 3-2 Expérience réalisée par Darcy pour valider sa loi empirique (d'après deMarsily 1986)

En rapport avec la figure 3.2, cette loi empirique établit que le débit d'eau qui passe à travers un substrat poreux peut être calculé par :

$$Q = -\frac{KA\Delta h}{L}$$

Éq. 2

où 'A' est l'aire de la section perpendiculaire à l'écoulement, 'L' est la longueur de matériau poreux traversé, 'Δh' est la perte de charge hydraulique entre les extrémités de 'L', et 'K' est une constante qui caractérise une propriété intrinsèque du milieu poreux parallèlement à l'écoulement, la conductivité hydraulique. L'équation 2 se lit comme suit : « Le débit d'un écoulement souterrain est proportionnel à la perte de charge hydraulique », (Brémond, 1965).

Toute solution du modèle analytique est construite en référence à un volume élémentaire de matériau poreux représentatif des propriétés du milieu dans lequel l'eau se déplace et pour lequel la loi de conservation de la masse vérifie qu'il n'y ait ni «apparition» ni «disparition» d'eau dans le système. La figure 3.3 présente le volume élémentaire de référence utilisé dans les démonstrations qui suivent :

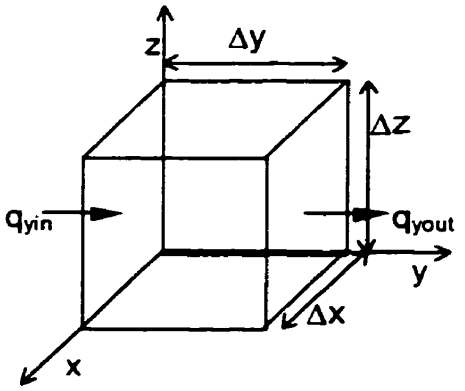

Figure 3-3 Le volume élémentaire de référence et quelques-uns unes de ses composantes (d'après Anderson et Woessner, 1992).

Les équations qui suivent, présentées ainsi par Anderson et Woessner (1992), ne sont valides que si l'on considère l'eau comme un fluide incompressible dont la densité n'est pas affectée significativement par les variations de pression et de température. Pour le volume élémentaire de la dernière figure, cette équation peut s'écrire comme suit :

$$\textit{Débit}_{\textit{sortant}} - \textit{Débit}_{\textit{entrant}} = \textit{Variation du volume stocké}$$

$$Q_{out} - Q_{in} = \text{Variation du volume stocké}$$

Éq. 3

On peut décomposer les débits entrant et sortant de l'équation 3 en leurs composantes orthogonales, selon les axes de référence et les vecteurs unitaires.

$$\text{Débit} = Q = q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

Les termes q_x, q_y et q_z sont des flux ayant la dimension d'une vitesse (L/T). Considérant le flux qui passe par les faces $\Delta x\Delta z$ du volume élémentaire, soit l'eau qui s'écoule parallèlement à l'axe y, le débit volumétrique (L^3/T) est donné par :

$$\left(q_{y_{out}} - q_{y_{in}}\right)\Delta x\Delta z$$

$$\frac{\left[\left(q_{y_{out}} - q_{y_{in}}\right)\Delta x\Delta y\Delta z\right]}{\Delta y}$$

$$\left(\frac{\partial q_y}{\partial y}\right)\Delta x\Delta y\Delta z$$

Procédant ainsi, on transforme l'équation 3 de façon à exprimer les flux selon chacun des axes. Cette équation devient alors :

$$\left(\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right)\Delta x\Delta y\Delta z = \text{Variation du volume stocké}$$

Cette dernière expression décrit un système fermé pour lequel aucune eau n'est ajoutée de l'extérieur, ni retirée. On doit donc y inclure un terme qui comptabilise les perturbations externes au régime d'écoulement. Ce terme permet de représenter, par exemple, les ajouts par irrigation et les retraits par pompage. Par convention, on note ce débit volumétrique '$R_s\Delta x\Delta y\Delta z$' (Anderson et Woessner, 1992). Étant intrinsèquement positif lorsqu'il s'agit d'un apport d'eau, on le soustrait au membre de gauche, ce qui donne :

$$\left(\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}-R_s\right)\Delta x\Delta y\Delta z = \text{Variation du volume stocké}$$

Éq. 4

Tout comme pour le modèle unicellulaire, le membre de droite de cette équation, soit la variation du volume stocké, est une fonction de la capacité spécifique d'emmagasinement 'S_s' du milieu poreux. Ce paramètre met en relation les variations de pression d'eau dans le volume élémentaire et la compressibilité du milieu poreux. La pression d'eau dans le volume élémentaire étant fonction de la colonne d'eau 'h' (piézométrie), on définit le coefficient d'emmagasinement comme étant le volume d'eau libéré 'ΔV' par rapport au volume élémentaire '$\Delta x\Delta y\Delta z$' pour une perte de charge 'Δh' unitaire (deMarsily, 1986). Par convention, on pose 'ΔV' intrinsèquement positif lorsque 'Δh' est négatif. Le coefficient d'emmagasinement est donc donné par :

$$S_s = -\frac{\Delta V}{\Delta h\Delta x\Delta y\Delta z}$$

La *'Variation du volume stocké'* de l'équation 4 exprime donc les fluctuations dans le temps du volume d'eau contenu dans le système, soit :

$$\frac{\Delta V}{\Delta t} = -\frac{S_s \Delta h}{\Delta t} \Delta x \Delta y \Delta z$$

Éq. 5

En combinant les équations 4 et 5 on obtient alors :

$$\left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} - R_s \right) \Delta x \Delta y \Delta z = -\frac{S_s \Delta h}{\Delta t} \Delta x \Delta y \Delta z$$

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} - R_s = -S_s \frac{\partial h}{\partial t}$$

Éq. 6

Pour pouvoir utiliser cette dernière équation, on doit exprimer les débits spécifiques 'q_x', 'q_y' et 'q_z' en fonction des caractéristiques du milieu d'écoulement à l'aide de la loi empirique de Darcy. Pour la composante en *'z'* du débit qui traverse le volume élémentaire, on peut alors écrire :

$$q_x = -K_x \frac{\partial h}{\partial x}$$

Les termes 'q_y' et 'q_z' prennent des formes similaires.

L'expression finale de l'équation gouvernante pour les écoulements d'eau souterraine est obtenue en remplaçant les débits spécifiques de l'équation 6. On obtient alors :

$$\frac{\partial\left(-K_x \frac{\partial h}{\partial x}\right)}{\partial x} + \frac{\partial\left(-K_y \frac{\partial h}{\partial y}\right)}{\partial y} + \frac{\partial\left(-K_z \frac{\partial h}{\partial z}\right)}{\partial z} - R_s = -S_s \frac{\partial h}{\partial t}$$

Éq. 7

$$\nabla \cdot (\mathbf{K} \nabla h) + R_s = S_s \frac{\partial h}{\partial t}$$

Éq. 8

Notons ici que l'équation 7 constitue une forme simplifiée de l'équation 8. Pour résoudre directement cette dernière équation, l'approche analytique introduit plusieurs simplifications. Dans un premier temps, elle suppose généralement que l'écoulement vertical de l'eau ne dépend pas des gradients de la surface piézométrique, autrement dit que les flux selon '*z*' sont indépendant des gradients hydrauliques dans le plan '*xy*' (René Therrien, 1995, communication personnelle). Cette première simplification permet d'intégrer l'équation 6 selon *'z'* entre *'0'* et *'h'*, ramenant le problème tridimensionnel à un problème bidimensionnel. Pour la composante en *'x'*, on obtient :

$$\int_0^h \left[\frac{\partial \left(-K_x \frac{\partial h}{\partial x} \right)}{\partial x} \right] dz = - \frac{\partial \left(\frac{\partial h}{\partial x} \int_0^h K_x dz \right)}{\partial x} = - \frac{\partial \left(K_x h \frac{\partial h}{\partial x} \right)}{\partial x}$$

Le terme en '*y*' de l'équation 7 est similaire à celui-ci. Le terme en *'z'* est quant à lui remplacé par une fonction *'L'* qui quantifie les transferts de masse verticaux vers le système et hors de ce dernier par delà une épaisseur *'b'* de matériaux poreux. Les valeurs spécifiques *'S_s'* et *'R_s'* sont elles aussi intégrées selon *'z'*, devenant des coefficients moyens par unité de surface, nommément *'S'* et *'R'* (Anderson et Woessner, 1992).

$$L = -K_z \frac{\Delta h}{b}$$

$$\frac{\partial \left(K_x h \frac{\partial h}{\partial x} \right)}{\partial x} + \frac{\partial \left(K_y h \frac{\partial h}{\partial y} \right)}{\partial y} = S \frac{\partial h}{\partial t} - R + L$$

Éq. 9

$$\nabla \cdot (\mathbf{K}h\nabla h) = S\frac{\partial h}{\partial t} - R + L$$

Éq. 10

Des simplifications additionnelles sont souvent introduites à ce stade, relatives quant à elles à la géométrie et l'état intrinsèque du domaine : le milieu est considéré inconditionnellement homogène et d'épaisseur constante. Par commodité, la transmissivité hydraulique *'T'* est introduite, intégrant la conductivité hydraulique **'K'** selon *'z'* sur l'épaisseur *'h'* de l'aquifère. L'équation 10 devient alors (d'après Anderson et Woessner, 1992) :

$$T\left[\nabla^2 h\right] = S\frac{\partial h}{\partial t} - R + L$$

Éq. 11

La grande majorité des solutions analytiques sont dérivées de cette dernière équation en ne considérant qu'une seule singularité, telle une rivière, une station de pompage, un barrage, etc. a géométrie réelle du domaine n'est pas considérée, pas plus que son hétérogénéité, les interactions entre les différentes singularités qui le définissent et les conditions effectives aux frontières. Il s'agit néanmoins d'un outil de modélisation plus souple que le modèle unicellulaire en ce qu'il permet d'évaluer avec exactitude la répartition de l'eau au voisinage d'une singularité. Pour les problèmes simples où les conditions réelles s'approchent des conditions idéales, les solutions obtenues sont pratiquement exactes.

Les situations rencontrées dans la pratique sont cependant rarement aussi simples que ce qui est supposé dans le modèle analytique. Il est clair que les limites du modèle sont rencontrées lorsque les conditions réelles tranchent radicalement des conditions idéales ou encore lorsque plusieurs singularités ponctuent un bassin et qu'une vue synoptique en est requise. Typiquement, un bassin présente une géométrie irrégulière, des paramètres hydrauliques qui varient dans l'espace et dans le temps, des frontières complexes et plusieurs ouvrages hydrauliques, bref un assemblage qui se trouve fort mal accommodé par les équations dérivées à l'aide de l'approche analytique. La mise en œuvre de ces solutions passe par un lissage de la complexité qui mène parfois à des aberrations (Boonstra et de Ridder, 1981). Un outil de

modélisation plus précis doit être utilisé pour en arriver à une solution aussi globale que celle du modèle unicellulaire et aussi exacte que celle du modèle analytique.

3.3 Le modèle distribué

Le modèle distribué offre une alternative aux modèles unicellulaire et analytique. Dérivé de ces derniers, il en garde la vision à la fois locale et globale. Le modèle distribué est constitué d'une multitude de tuiles jointives (cellules) recouvrant la totalité du bassin. Il évalue le bilan hydrique pour chaque cellule de la même façon que le modèle unicellulaire et les transferts de masse entre cellules voisines avec l'équation de Darcy, comme le modèle analytique. La simulation passe alors par la résolution d'un système d'équations linéaires donnant une solution approchée pour chacune des cellules du modèle.

Le modèle distribué schématise le système d'écoulement par une approche conceptuelle, le décomposant en ses éléments constitutifs fondamentaux que sont :

- les frontières externes ;
- les barrières internes ;
- les sites d'ajouts / de retraits de masse ;
- le paramètre de conductivité hydraulique / transmissivité ;
- le coefficient d'emmagasinement ;
- la piézométrie.

Les composantes ci-dessus énumérées sont plus bas décrites en détail. Il est à noter que le modèle distribué tient son nom du fait qu'il intègre la distribution spatio-temporelle de tous ces éléments dans une simulation.

3.3.1 Frontière externe.

La frontière externe d'un domaine d'écoulement est le lieu géométrique qui en délimite la surface. La piézométrie ou encore son gradient y est généralement connu. On observe plusieurs types de frontières externes. Une ligne de partage des eaux constitue, comme il a été vu plus haut, une frontière imperméable, c'est-à-dire un lieu géométrique où le gradient piézométrique normal est nul. De la même façon, le contact entre un sol poreux et une

formation géologique imperméable constitue une frontière à débit nul, i.e. où le gradient hydraulique est nul (voir la figure 3.4).

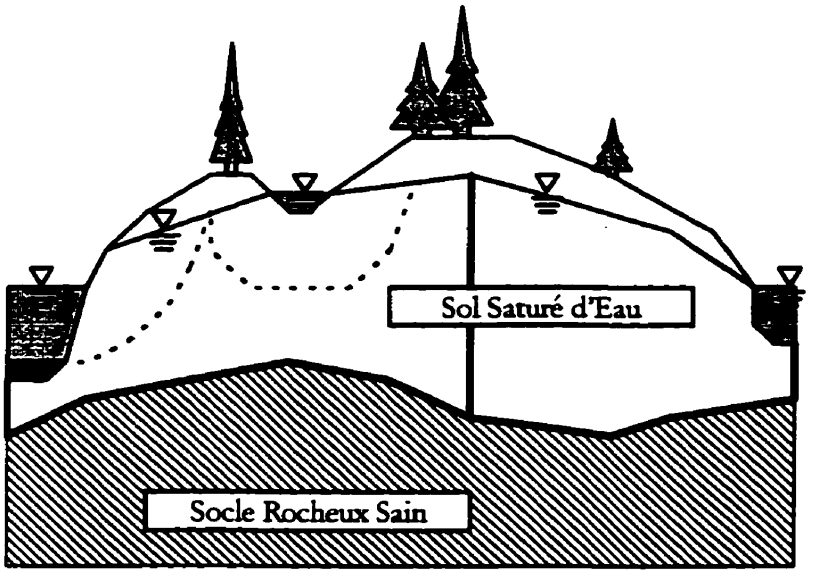

Figure 3-4 Système d'écoulement en coupe. Les traits pleins symbolisent les grandes lignes de partage. Les traits pointillés indiquent les limites entre zones de drainages contiguës. (d'après Anderson et Woessner, 1992)

Il existe aussi des frontières où la piézométrie est fixe et connue. Ces frontières correspondent la plupart du temps aux zones de contact entre le sol et les eaux de surface, comme les rives d'un lac, les berges d'une rivière, ou encore une résurgence, comme le montre la figure 3.4.

Finalement, on trouve aussi des frontières dites «mixtes» où prend place une percolation provoquée par la poussée gravitaire d'une colonne d'eau. Elles séparent généralement le fond d'un lac (rivière, drain, etc.) de la réserve souterraine. Le fond de la rivière qui borde l'ouest du système d'écoulement de la figure 3.4 peut être considéré une frontière de ce type.

Chacune de ces frontières correspond à un phénomène physique particulier. À un lieu géométrique est donc associé un comportement exprimé sous la forme d'une équation mathématique :

$$h = cte \ : \ \text{Frontière à charge imposée}$$

$$\frac{\partial h}{\partial n} = cte \ : \ \text{Frontière à débit fixe}$$

$$h + \alpha \frac{\partial h}{\partial n} = cte \text{ : Frontière mixte}$$

où '*cte*' et '*a*' sont les valeurs constantes qui caractérisent les frontières, et '*n*' est la normale externe à ces frontières.

Certains auteurs font une distinction entre «frontières physiques» et «frontières hydrauliques» (Anderson et Woessner, 1992). Selon ces derniers, les frontières physiques du bassin sont les zones de contact qui séparent les formations aquifères des formations géologiques imperméables ou des eaux de surface. Les frontières hydrauliques résultent quant à elles de conditions particulières qui peuvent évoluer rapidement dans le temps, comme les lignes de partages des eaux et les lignes de courants.

3.3.2 Barrière interne.

Les barrières internes sont les lieux géométriques du bassin caractérisés par les mêmes phénomènes physiques que ceux rencontrés aux frontières externes, à la seule différence près qu'ils ne circonscrivent pas le domaine d'écoulement que l'on veut modéliser.

Les cours d'eau, les lacs et les drains sont en effet des singularités du bassin qui en modulent le régime d'écoulement, de la même façon que les frontières externes. Outre leur localisation et leur relation topologique en rapport au bassin, l'interprétation mathématique donnée aux barrières internes est identique à celle des frontières externes.

3.3.3 Site d'ajout / de retrait de masse.

Plusieurs ouvrages hydrauliques visent à alimenter en eau des populations humaines ou animales, ou encore des industries. D'autres installations servent à recharger ces mêmes réserves par infiltration, percolation ou par injection forcée. Ces sollicitations ont un effet sensible sur le régime d'écoulement du bassin qu'il convient d'inclure dans une simulation.

Le pompage et l'injection forcée d'eau dans le sous-sol constituent de bons exemples de sollicitations concentrées. Les pluies et l'irrigation des terres sont pour leur part considérées

comme des sollicitations réparties. Ces sollicitations, concentrées et réparties, modifient le régime d'écoulement et sont interprétées comme des cas de sous-espèce des barrières internes.

3.3.4 Conductivité hydraulique / transmissivité.

Les propriétés physiques du milieu d'écoulement modulent la répartition et le mouvement de l'eau souterraine. La conductivité hydraulique (L/T), qui exprime la faculté d'un milieu poreux à se laisser traverser par un fluide soumis à une certaine charge (Brémond, 1965), est une des propriétés physiques les plus importantes d'un système hydrogéologique. Des matériaux opposent une résistance plus grande que d'autres au passage de l'eau, et il existe toute une gamme de conductivités hydrauliques, depuis les milieux pratiquement imperméables (granite sain), jusqu'aux matériaux très perméables (gravier lâche).

Les informations relatives à la conductivité hydraulique se présentent généralement sous forme de levés ponctuels dont les mesures se regroupent à l'intérieur d'une plage de valeurs plus ou moins étendue pour une même formation géologique. Chacune de ces mesures ponctuelles ne reflète pas nécessairement une grandeur absolue étant donné, il faut l'admettre, le manque précision des valeurs dérivées des mesures de sondage. Elles doivent être plutôt interprétées ensemble comme une population statistique.

La transmissivité *'T'* (L^2/T) est une abstraction mathématique qui intègre toutes les valeurs de la conductivité hydraulique *'K'* rencontrées sur l'épaisseur verticale *'b'* du milieu d'écoulement. Ce paramètre est intéressant puisque dans les faits, rien ne différencie une lame d'eau très épaisse circulant dans un matériau presque imperméable d'une seconde très mince contenue dans un horizon perméable. La transmissivité peut donc être interprétée plus largement comme étant l'aptitude d'une formation aquifère à véhiculer l'eau. Pour la suite de ce travail, nous considérerons que ces valeurs sont les mêmes pour tous les axes, i.e. que le milieu poreux est isotrope.

3.3.5 Coefficient d'emmagasinement.

Le coefficient d'emmagasinement spécifique (L^{-1}) est une caractéristique hydraulique intrinsèque du milieu d'écoulement qui établit une relation entre le bilan des entrées/sorties et

les variations de la piézométrie dans le temps. Cette grandeur reflète à la fois la compressibilité du fluide, du squelette granulaire et des grains eux-mêmes. Elle établit la proportion d'eau mobile en fonction du volume de pores saturés et du volume d'eau qui y est fixée par capillarité, affinité chimique ou tension moléculaire.

3.3.6 Piézométrie.

La piézométrie *'h'* (L), aussi connue sous le nom de charge hydraulique, d'une formation aquifère est l'élévation à laquelle se trouve le niveau de la nappe d'eau. Plus exactement, la surface piézométrique est le profil topographique défini par l'horizon saturé d'une nappe d'eau souterraine.

Cette surface est l'image qui résulte de l'effet combiné de tous les paramètres énoncés plus haut et reflète l'état dynamique d'un bassin. Dans la pratique, la charge hydraulique est la composante d'un système d'écoulement que l'on tente généralement de dériver synthétiquement à l'aide de modèles mathématiques et de solutions numériques.

Sur le plan en coupe de la figure 3.4, la surface libre est symbolisée par un petit triangle pointé vers le bas et dont la pointe touche la surface de l'horizon saturé.

4 IFDM : Une perspective historique et mathématique.

Dans ce chapitre nous dressons un portrait de la méthode des différences finies intégrées (IFDM) dans une perspective à la fois historique et mathématique. Il ne s'agit pas d'un historique détaillé de la simulation numérique en hydrogéologie, mais d'une esquisse du chemin particulier qui relie chacune des étapes ayant mené au développement de la IFDM.

4.1 MacNeal.

Dans les années 1940-1950, les exercices de simulation étaient principalement fondés sur la théorie du calcul numérique par différences finies. Un modèle était alors réalisé à l'aide de résistances électriques et de condensateurs, tous reliés à des clous disposés selon une maille régulière sur une planche, le résultat étant lu à l'aide d'un voltmètre. On s'en doute, une telle manière de faire venait avec un lot de problèmes qui compliquaient grandement la tâche des spécialistes.

Parmi toutes les complications décrites en long et en large dans la littérature de l'époque, on retient que la principale venait de l'utilisation d'une subdivision régulière uniforme du domaine d'intérêt, généralement en tuiles carrées ou rectangulaires. L'utilisation de ce modèle géométrique empêchait en effet les nœuds internes de correspondre exactement à la position des singularités du domaine et de la même façon, empêchait l'alignement des nœuds externes de correspondre à la position exacte des frontières (c.f. figure 4.1).

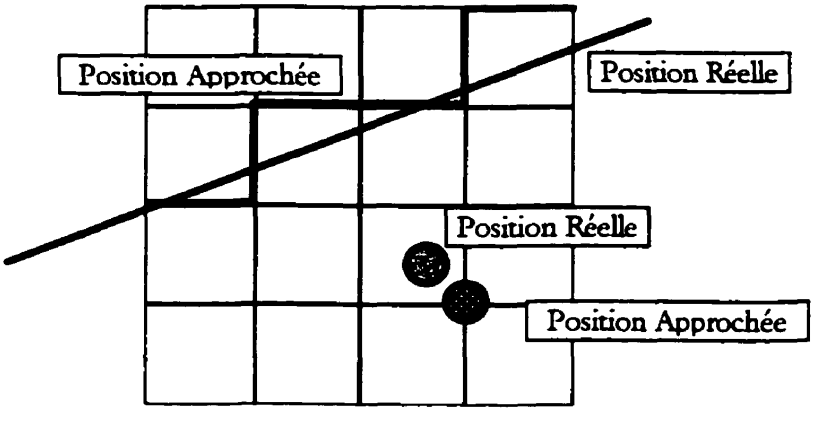

Figure 4-1 Portion d'un maillage régulier montrant clairement l'imprécision introduite dans le modèle par la représentation des singularités et des frontières du domaine.

Pour obtenir une solution précise à ces endroits stratégiques, les chercheurs ont dû faire preuve d'ingéniosité et développer des fonctions d'interpolation qui leur permettaient d'y estimer une valeur. Plusieurs articles traitant de ces fonctions d'interpolation paraissent alors (Southwell, 1946 et Emmons, 1944) sans qu'une seule ne soit validée par une comparaison entre les résultats obtenus et la réalité physique observée (MacNeal, 1953).

Certains préconisaient comme alternative l'utilisation de maillages très fins. Bien qu'en théorie il s'agisse d'une idée tout à fait valable, en pratique il en était tout autre. Considérant que le raffinement de la maille entraîne une forte augmentation des coûts de réalisation en pièces (résistances et condensateurs) et en temps d'installation, bien rares étaient les équipes qui avaient les moyens de se payer le montage qui leur aurait permis d'obtenir la précision souhaitée. Limitées par la taille des enveloppes budgétaires mises à leur disposition, la grande majorité des laboratoires devaient donc faire un choix : utiliser des fonctions d'interpolation jamais encore validées, ou alors trouver l'équilibre parfait entre le raffinement du maillage et les dépenses admises.

C'est en tentant de trouver une réponse à cet épineux problème que MacNeal (1953) entreprend une démarche qui le conduit bientôt à explorer le potentiel des triangulations de points comme solution de rechange aux mailles régulières comme support aux exercices de simulation. Il en arrive à démontrer que pour les problèmes physiques bidimensionnels, les coefficients des termes de l'équation d'un modèle mathématique ne résultent que de l'agencement géométrique des nœuds dans un réseau triangulaire.

MacNeal est en fait le premier physicien à avoir jamais utilisé une subdivision irrégulière de l'espace dans l'application des calculs par différences finies pour une simulation. Il était désormais possible de faire correspondre les singularités et les frontières du domaine avec les nœuds du modèle. Les maillages irréguliers à densité variable, comme celui de la figure 4.2, éliminaient du même coup l'utilisation des fonctions d'interpolation, tout comme les compromis à faire en regard des dépenses admises.

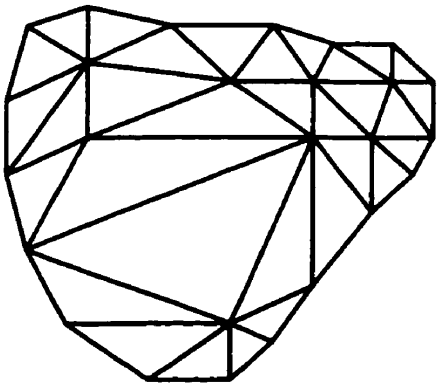

Figure 4-2 Maillage irrégulier à densité variable. La taille des tuiles triangulaire est ajustée de manière à ce que les frontières et singularités du modèle correspondent aux positions réelles.

4.2 Dussinberre.

Dussinberre (1961) donne par la suite une démonstration rigoureuse de l'équivalence géométrique entre les tuiles orthogonales régulières des différences finies classiques (FDM) et celles proposées quelques années plus tôt par MacNeal (1953).

Il en souligne au passage les avantages et propose son utilisation pour la résolution de tous les problèmes de transferts de chaleur par conduction pour les domaines aux frontières irrégulières.

4.3 *Tyson et Weber.*

Tyson et Weber (1963) transposent la méthode de MacNeal, développée pour les transferts de chaleurs, à la résolution des problèmes d'écoulement des eaux souterraines. S'appuyant sur le fait que les équations du transfert de chaleur par conduction sont conceptuellement semblables à celles du transfert de masse, ils réussissent à démontrer que le modèle polygonal de MacNeal est aussi applicable en hydrogéologie (Tyson et Weber, 1963).

Dès l'année suivante, ils appliquent leur propre version du modèle de MacNeal à l'étude du régime d'écoulement des eaux souterraines de la Californie (Tyson et Weber, 1964). Ils obtiennent des résultats significatifs qui leur permettent non seulement de valider l'approche, mais aussi d'émettre quelques recommandations aux autorités locales en matière d'aménagement des ressources hydriques.

4.4 Thomas.

Thomas (1973) expose les détails du modèle hydrogéologique polygonal développé par Tyson et Weber en faisant une démonstration mathématique très complète de la résolution de l'équation pour les problèmes bidimensionnels.

Dans ce document, il décrit la configuration optimale des représentations géométriques pour l'application de la méthode comme étant une couverture polygonale constituée de la jonction des médianes perpendiculaires aux arcs d'une triangulation. Sans définir clairement ce qu'il entend par «triangulation», il prend tout de même soin de spécifier qu'aucun arc ne doit en recouper un second, et suggère l'utilisation d'une triangulation qui minimise le nombre d'angles obtus pour des raisons de stabilité numérique.

Ainsi donc, dans ce genre de construction chaque vertex de la triangulation se trouve entouré d'une portion de l'espace qui lui est exclusive, portion sur laquelle les bilans de masse sont évalués en cours de simulation. Une couverture polygonale de ce type est donnée à la figure 4.3.

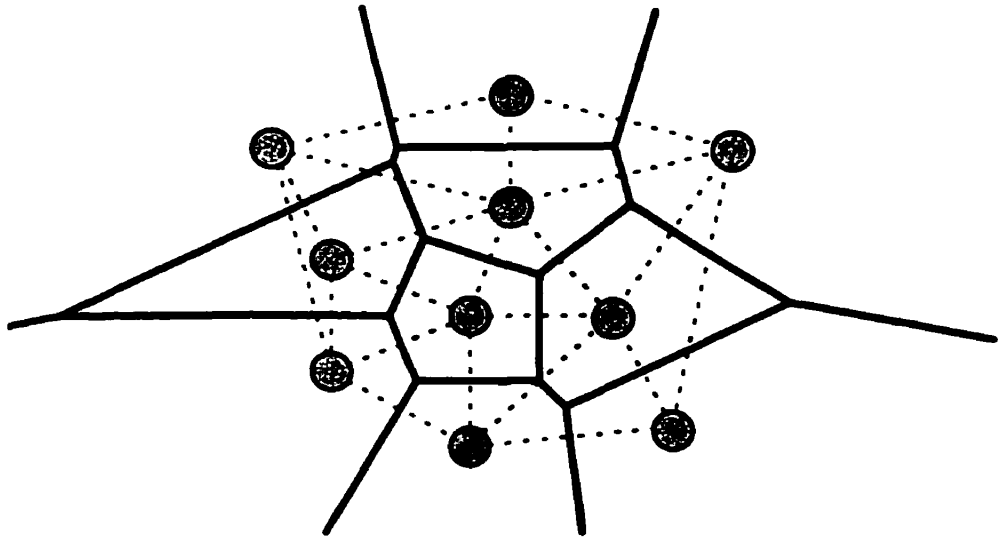

Figure 4-3 Couverture polygonale telle que décrite par Thomas.

L'équation des écoulements souterrains, qui décrit les transferts en termes de dérivées partielles, est remplacée par un système équivalent d'équations aux différences finies dont la projection sur le modèle géométrique conduit à la formation d'un système d'équations

linéaires où chaque équation du système correspond au bilan hydrique d'une et une seule cellule du modèle polygonal. Suivant l'approche décrite par Thomas, pour le nœud '*P*' entouré de '*n*' nœuds '*i*' voisins, cette équation prend la forme suivante :

$$\sum_{i=1}^{i=n} T_{iP} \frac{(h_i - h_P)}{X_{iP}} \times B_{iP} = A_P S_P \frac{dh_P}{dt} - A_P R_P + A_P L_P$$

Éq. 12

Paramètre	**Description**	**Dimension**
T_{iP}	Transmissivité entre les nœuds '*i*' et '*P*'	$M^3 T^{-1} L^{-1}$
h_x	Piézométrie au nœud '*x*'	L
X_{iP}	Distance entre les nœuds '*i*' et '*P*'	L
B_{iP}	Longueur de l'interface commune aux nœuds '*i*' et '*P*'	L
A_P	Aire de la cellule délimitée par les interfaces de '*P*'	L^2
S_P	Coefficient d'emmagasinement de la cellule '*P*'	n.a.
$\frac{dh_P}{dt}$	Variation de la charge au nœud '*P*' dans le temps	LT^{-1}
R_P	Ajout/retrait d'eau au prisme associé au nœud '*P*'	$M^3T^{-1}L^{-2}$
L_P	Ajout/retrait d'eau selon z sur l'aire de la cellule '*P*'	$M^3T^{-1}L^{-2}$

Le membre de gauche de l'équation 12 correspond en fait à la sommation des débits horizontaux qui prennent place entre la cellule '*P*' et les '*n*' cellules '*i*' voisines. Le premier terme du membre de droite représente quant à lui le taux d'emmagasinement de l'eau dans la

cellule *'P'*. Les termes restants du membre de droite comptabilisent quant à eux les débits selon *'z'* qui entrent et qui sortent de la cellule *'P'*. La figure 4.4 illustre la distribution des paramètres dans le modèle polygonal de Tyson et Weber.

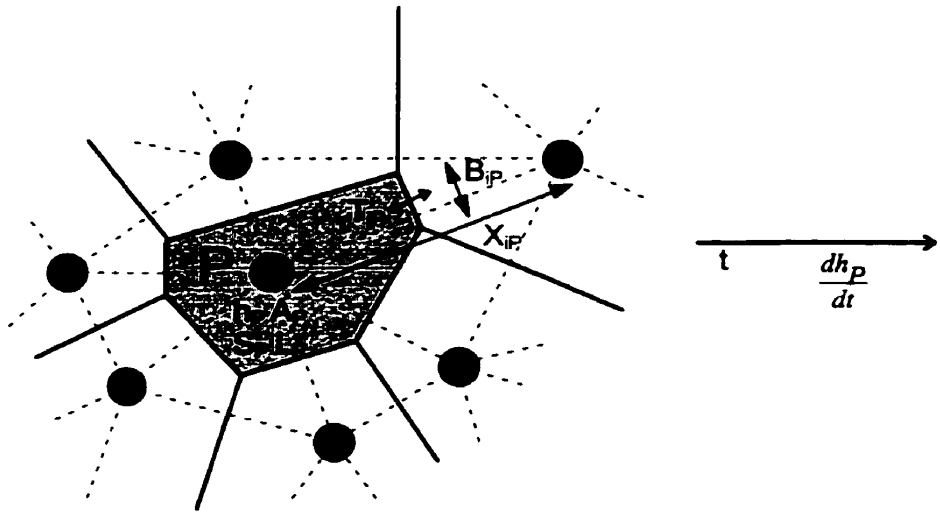

Figure 4-4 Distribution des paramètres géométriques et physiques sur le modèle polygonal.

Au moment où le modèle polygonal a été développé, et jusqu'à récemment, la puissance et la convivialité des ordinateurs rendaient les tâches de saisie et de structuration des données tellement lourdes qu'il valait souvent mieux en faire des étapes préparatoires entièrement réalisées à la main. Les triangles devaient tous être construits à l'avance, de même que les relations d'adjacence entre les cellules du modèle, le calcul de la longueur des faces communes, etc. Un changement à la structure polygonale, même mineur, nécessitait la reconstruction manuelle entière des tables d'attributs et d'adjacences. Ce mode de gestion de l'information a fort probablement contribué à rendre le modèle polygonal de Thomas peu attrayant malgré les avantages manifeste qu'il présentait sur la FDM.

4.5 Narasimhan et Witherspoon.

Reprenant quelques années plus tard les travaux de leurs prédécesseurs, Narasimhan et Witherspoon décrivent avec rigueur le fondement théorique du modèle polygonal en hydrogéologie (1976, 1977, 1978). Jusque là en effet, le développement du modèle polygonal en hydrogéologie ne s'appuyait que sur la similitude conceptuelle qui existe entre les transferts de chaleur par conduction et les écoulements d'eau souterraine.

La démonstration mathématique très robuste qu'ils produisent alors définit en termes clairs tant les aspects algébriques et numériques du modèle polygonal que les aspects géométriques liés à sa construction. Ils valident aussi pour la première fois les solutions du modèle en les comparant aux résultats du modèle analytique. Ils élargissent finalement le cadre d'application du modèle pour lui permettre de résoudre les problèmes d'écoulements tridimensionnels. Ils baptisent ce modèle numérique «*Integrated Finite Differences Method*» (IFDM).

La suite de ce chapitre donne une explication mathématique de la IFDM telle que définie par Narasimhan et Witherspoon (1977, 1978). Soit le domaine '*D*' subdivisé en une multitude de petites cellules '*V*' sur lesquelles les bilans de masse seront évalués numériquement et dont les interfaces entre voisins sont perpendiculaires aux droites qui les relient. On intègre l'équation 8 sur un volume '*V*' de la même nature que celui présenté sur la figure 4.5.

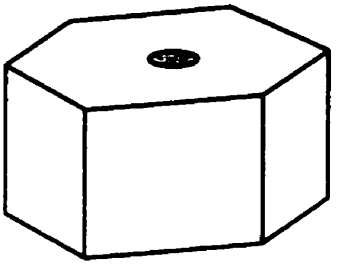

Figure 4-5 La cellule 'V' en 2D et son volume associé, une partie du domaine d'écoulement.

$$\int_V \left(\nabla \cdot (\mathbf{K}\nabla h) + R_s\right) dV = \int_V \left(S_s \frac{\partial h}{\partial t}\right) dV$$

$$\int_V \nabla \cdot (\mathbf{K}\nabla h) dV + \int_V R_s dV = \frac{\partial}{\partial t} \int_V (S_s h) dV$$

$$\int_V \nabla \cdot (\mathbf{K}\nabla h) dV + R_s V = S_s V \frac{\partial h}{\partial t}$$

Le premier terme du membre de gauche de cette équation est transformé en une intégrale de surface par le biais du théorème de Stoke, le vecteur '*n*' étant la normale externe à la surface '*S*' qui borne le volume '*V*'. Soit :

$$\int_S (\mathbf{K}\nabla h \cdot \mathbf{n}) dS + R_s V = S_s V \frac{\partial h}{\partial t}$$

Éq. 13

Le membre de gauche de l'équation 13 fait la sommation des flux passant perpendiculairement aux faces *'S'* du volume *'V'* et mesure donc le taux d'accumulation de l'eau dans *'V'*. Le membre de droite convertit quant à lui ce taux d'accumulation en une variation correspondante de la piézométrie *'h'* pour *'V'* dans le temps. L'approximation du gradient de *'h'* par différences finies transforme l'équation 13 en une seconde, parente à celle développée par Tyson et Weber (1963). Soit :

$$\sum_{i=1}^{i=n} K_{iP} \frac{(h_i - h_P)}{X_{iP}} \times F_{iP} = V_P S_{sP} \frac{dh_P}{dt} - V_P R_{sP}$$

Éq. 14

où les variables *'K_{iP}'*, *'X_{iP}'* et *'F_{iP}'* sont respectivement la conductivité hydraulique moyenne, la distance et la surface de contact entre les cellules *'i'* et *'P'*. Ce dernier terme peut être calculé de deux façons, soit en considérant la piézométrie moyenne entre *'i'* et *'P'* pour les écoulements non confinés ou toute l'épaisseur de l'horizon saturé pour les écoulements confinés. Les paramètres *'V_P'*, *'S_{sP}'* et *'R_{sP}'* sont dans l'ordre : le volume saturé de la cellule, son coefficient spécifique d'emmagasinement et le débit volumétrique d'ajout/retrait qui l'affecte.

Au prix d'une complexité mathématique relativement faible, le modèle distribué de Narasimhan et Witherspoon permet de simuler le comportement de l'eau souterraine dans des milieux hétérogènes à géométrie complexe. La preuve mathématique rigoureuse qui en est faite de même que sa validation pour les problèmes tridimensionnels en font un outil de simulation robuste et très flexible.

5 Modèle de données spatiales Voronoï.

Nous présentons dans ce chapitre un aperçu de la méthodologie Voronoï telle qu'elle se voit appliquée au domaine des Systèmes d'Information à Référence Spatiale (SIRS). Le modèle de l'espace Voronoï se présente comme un complément aux modèles de l'espace jusqu'à présent utilisés dans les SIRS, réconciliant dans une certaine mesure les approches matricielles et vectorielles. Il offre un support ubiquiste à l'information descriptive portant tant sur les champs continus (distribution des températures, des précipitations, etc.) que sur les entités physiques (routes, constructions, peuplements forestiers, etc.) (Gold et Edwards, 1993). Les concepts introduits à la géomatique par l'approche Voronoï donnent un sens nouveau à la notion de bases de données géographiques – qui, encore aujourd'hui, sont généralement perçues comme des documents statiques dont l'information correspond à un arrêt sur image de la réalité physique. La fabrique intime du modèle Voronoï lui permet de représenter des espaces géographiques dynamiques, peuplés d'entités représentant des phénomènes évolutifs.

Au cœur du modèle se trouve la définition explicite des relations de voisinage entre tous les objets contenus dans un espace. Ces relations d'adjacence constituent un réseau fortement connecté d'arcs que l'on peut défiler systématiquement ou traverser sélectivement pour joindre sur une carte digitale le point A au point B. Ce modèle structure les données de proche en proche, faisant en sorte que toute modification apportée à sa structure ne peut avoir qu'un effet local. Dans une perpective applicative, le diagramme Voronoï constitue un outil précieux puisqu'il s'agit d'une structure qui supporte le développement de solutions liées aussi bien aux graphes et aux arbres, qu'aux espaces rasteurs : chaque cellule Voronoï peut en effet être considérée comme un élément constitutif d'une couverture de type rasteur, comme un cycle dans un graphe ou un embranchement dans un arbre.

En termes admis, le diagramme Voronoï est définit comme une structure géométrique recouvrant l'espace de tuiles jointives toutes générées par un nœud, et dont la particularité tient au fait que tous les points d'une tuile sont plus près du nœud qui l'a généré que de tout autre. Pour un ensemble de nœuds P_i dans le plan, l'expression du concept de proximité

résulte en une couverture polygonale associant une tuile à chacun des nœuds P_I, comme le montre la figure 5.1.

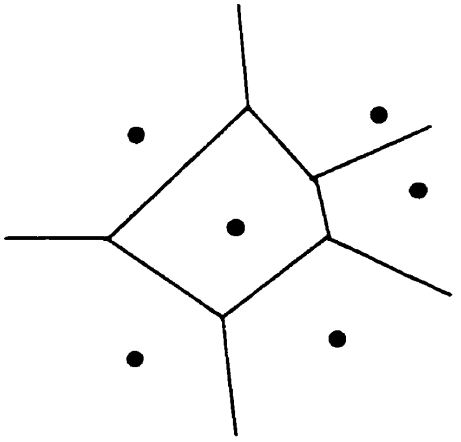

Figure 5-1 Diagramme Voronoï de points dans le plan.

Les mécanismes liés à la construction et au maintien de cette structure se rapprochent conceptuellement des principes de navigation, simplifiant le rapport entre l'usager, ses données spatiales et les opérations qu'il leur applique. Un SIRS exploitant le modèle Voronoï présente un mode de fonctionnement voisin de celui que nous utilisons d'instinct – notre système de «navigation intégrée» - permettant aux usagers de se projeter parmi les entités de la carte, d'interagir directement avec celles-ci d'une façon tout à fait naturelle. Cette approche est qualifiée de «navale» par opposition à l'approche «aérienne» des modèles conventionnels où l'usager peut voir l'information sans pouvoir interagir simplement avec cette dernière (Gold, 1994).

5.1 Diagramme Voronoï de points sur la droite.

Dans sa forme la plus simple, considérons la construction incrémentale d'un diagramme Voronoï de points dans un espace à une dimension. Bien qu'il paraisse trivial d'en faire une description approfondie, cet exercice fait ressortir certaines propriétés fondamentales que l'on observe aussi sur des diagrammes enchâssés dans des espaces multidimensionnels.

Les générateurs sont ici représentés par un ensemble de positions le long d'une droite. L'insertion de chacun de ces nœuds dans le diagramme induit la création de nouvelles

relations d'adjacence spatiale et la destruction de certaines autres. Le scénario de la figure 5.2 illustre le phénomène.

Figure 5-2 Diagramme Voronoï 1D où un vertex additionnel est ajouté. Les traits pointillés symbolisent les interfaces Voronoï.

Comme on le constate, juste avant l'insertion d'un nouvel élément, le diagramme Voronoï est valide, les interfaces séparant les paires de nœuds voisins ayant été convenablement construites jusque là. Il s'agit ici d'une caractéristique fondamentale que doit posséder tout algorithme de construction incrémentale.

Une fois la position du nœud 6 localisée dans le diagramme, deux nouvelles relations d'adjacence, 2-6 et 6-8, sont construites et une dernière détruite, soit 2-8. Cette construction du diagramme dans un espace à une dimension correspond dans les faits à la création d'une liste doublement chaînée. Au sens large, on peut donc interpréter les diagrammes Voronoï d'espaces supérieurs comme une généralisation du concept de la liste chaînée, soit une structure de données abstraite servant de conteneur à un ensemble de données occupant un

espace multidimensionnel, supportant l'insertion et la délétion de nœuds, et sur laquelle différentes stratégies de défilement peuvent être développées.

5.2 Le diagramme Voronoï de points dans le plan.

Sans minimiser l'importance des listes chaînées, leur généralisation pour des espaces multidimensionnels constitue un sujet de recherche fascinant traité par bon nombre de spécialistes de la géométrie algorithmique. Et la géomatique y trouve son compte puisqu'une variété de problèmes qu'elle étudie puise des éléments de solution au cas particulier de diagramme de *'n'* positions dans le plan. Nous décrirons donc ici brièvement le diagramme Voronoï de points dans le plan en utilisant une mise en situation fort simple, donnant au passage un algorithme incrémentale pour sa construction.

5.2.1 Le diagramme Voronoï bidimensionnel dans le quotidien.

Pour donner une image au concept abstrait de diagramme Voronoï de points dans le plan, supposons le scénario suivant. Une équipe de scientifiques trouve un champ de broussailles parfaitement plat et s'y rend par une journée sèche et sans vent. Les membres de l'équipe allument simultanément une série de foyers d'incendie et laissent le feu s'y propager librement. Étant donnée l'homogénéité des broussailles et l'absence de vent et d'humidité, le feu se déplace uniformément, à vitesse constante.

Ils observent que les fronts incendiaires immédiatement voisins s'éteignent lorsqu'ils se touchent : les flammes ont disparues le long de lignes formant un polygone autour de chacun des foyers. Ils observent que les flammes se sont ultimement éteintes sur des jonctions triples, sommets partagés par trois polygones distincts. Faisant une analyse du patron des crêtes d'extinction, ils découvrent alors un assemblage de tuiles toutes convexes et parfaitement imbriquées les unes aux autres - structure en tout point identique au diagramme Voronoï des foyers d'incendie.

L'observation attentive des données recueillies lors de cette expérience fait ressortir le fait que chacune des frontières d'extinction entre foyers voisins est en fait la bissectrice perpendiculaire à la droite qui relie des foyers voisins, comme le montre la figure 5.3. Le

périmètre d'une tuile est composé d'arcs tous perpendiculaires aux segments qui relient les foyers immédiatement voisins.

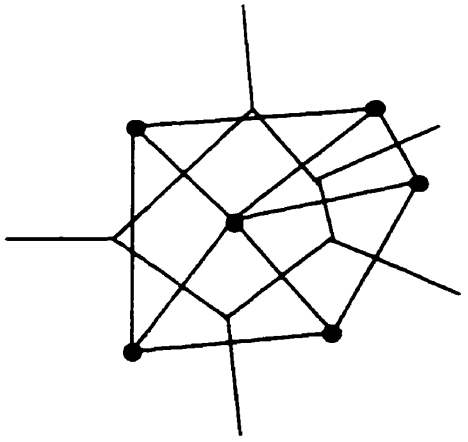

Figure 5-3 Diagramme Voronoï de points dans le plan et triangulation Delaunay correspondante.

L'assemblage des arcs qui relient les foyers voisins forme une triangulation dont tous les triangles sont mutuellement exclusifs. Le feu s'étant déplacé à vitesse constante, les jonctions triples où se sont éteintes les dernières flammes correspondent aux lieux géométriques équidistants aux trois sommets de chacun de ces triangles, soit les centres des cercles passant par les sommets de ces triangles (c.f. figure 5.4). Étant donnée l'isotropie des paramètres contrôlant le déplacement des fronts incendiaires, ces cercles ne peuvent contenir aucun autre foyer incendiaire.

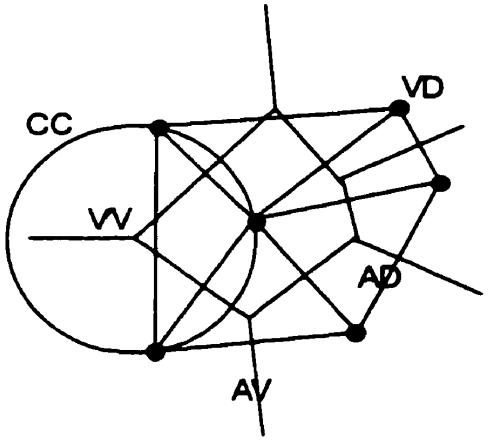

Figure 5-4 Un cercle passant par les sommets d'un triangle et ayant une jonction triple en son centre dans un diagramme Voronoï de points dans le plan.

Nous référant toujours à la figure 5.4, pour employer les termes techniques utilisés par les spécialistes de la question, nous considérerons pour le reste du document que :

- les foyers d'incendie sont des vertex Delaunay (VD) ;
- les arêtes de triangles sont des arcs Delaunay (AD) ;
- les triangles sont des simplexes Delaunay ;
- les cercles circonscrits sont des circumcercles (CC) ;
- les jonctions triples, les centres des cercles circonscrits, sont des vertex Voronoï (VV) ;
- les arêtes du périmètre des cellules polygonales entourant les foyers sont des arcs Voronoï (AV) ;
- les cellules polygonales entourant les foyers sont des polygones Voronoï.

L'analyse des observations relevées lors de cette expérience met en relief une des caractéristiques fondamentales du diagramme Voronoï dans le plan, à savoir que les sommets des polygones Voronoï correspondent aux centres des circumcercles. Cette seule propriété permet de définir une triangulation Delaunay, elle donne la clé permettant de construire une triangulation synthétique de laquelle peut être aisément dérivé le diagramme Voronoï (Guibas et Stolfi, 1985)

Comme nous l'avons énoncé plus tôt, l'ajout d'un vertex Delaunay au diagramme Voronoï ne peut avoir qu'un effet local sur sa structure. Seuls les simplexes dont le circumcercle contient le nouveau vertex Delaunay se verront modifiés. Inversement, la suppression d'un vertex Delaunay ne modifie les relations d'adjacence que dans le voisinage immédiat de l'opération, menant à la définition de nouveaux triangles dont les circumcercles incluent la position du vertex retiré.

Ce caractère local des opérations d'insertion et de suppression sur la configuration topologique du diagramme est une caractéristique intéressante pour des applications nécessitant de fréquentes modifications de cette nature, comme c'est le cas lors de la construction des bases de données géographiques. Cette caractéristique permet de maintenir la structure du diagramme Voronoï et d'en garantir l'intégrité géométrique et topologique en tout temps (Gold, 1990)

5.2.2 Un algorithme incrémentale pour la génération de diagramme Voronoï 2D de points.

Exploitant la propriété du circumcercle, de nombreux algorithmes de génération du diagramme Voronoï de points dans le plan ont été développés dans la dernière décennie. Ils mettent tous en jeu les mêmes mécanismes topologiques et les même tests géométriques. Les différences majeures entre les principales techniques apparaissent plutôt au niveau des pré-conditions, des séquences d'opérations et des structures de données. La méthode incrémentale de génération et de maintien du diagramme Voronoï se présente comme une des plus simples pour une complexité algorithmique randomisée de O($n$log$n$) en temps et de O(n) en espace.

Cet algorithme porte le nom de Bowyer-Watson (Chew et al, 1997). Il débute par une phase d'initialisation pendant laquelle est généré un «énorme» triangle destiné à contenir en son intérieur tous les vertex Delaunay qui seront considérés ultérieurement. Les vertex sont ensuite séquentiellement insérés dans le diagramme, un à un.

Cette procédure d'insertion nécessite l'identification puis l'élimination du triangle qui contient le nouveau vertex - ce triangle peut être éliminé sans autre test car le circumcercle qui lui est associé contient assurément le nouveau vertex.

Les triangles voisins à celui tout juste éliminé sont ensuite visités et eux aussi détruits dans l'éventualité où leur circumcercle contient le nouveau vertex. Les voisins des triangles détruits sont eux aussi visités de la même manière. Lorsque tous les triangles susceptibles d'être éliminés ont été défilés, de nouveau triangles sont générés, joignant les sommets de la cavité entourant le nouveau vertex à ce dernier. La figure 5.5 illustre ce mécanisme.

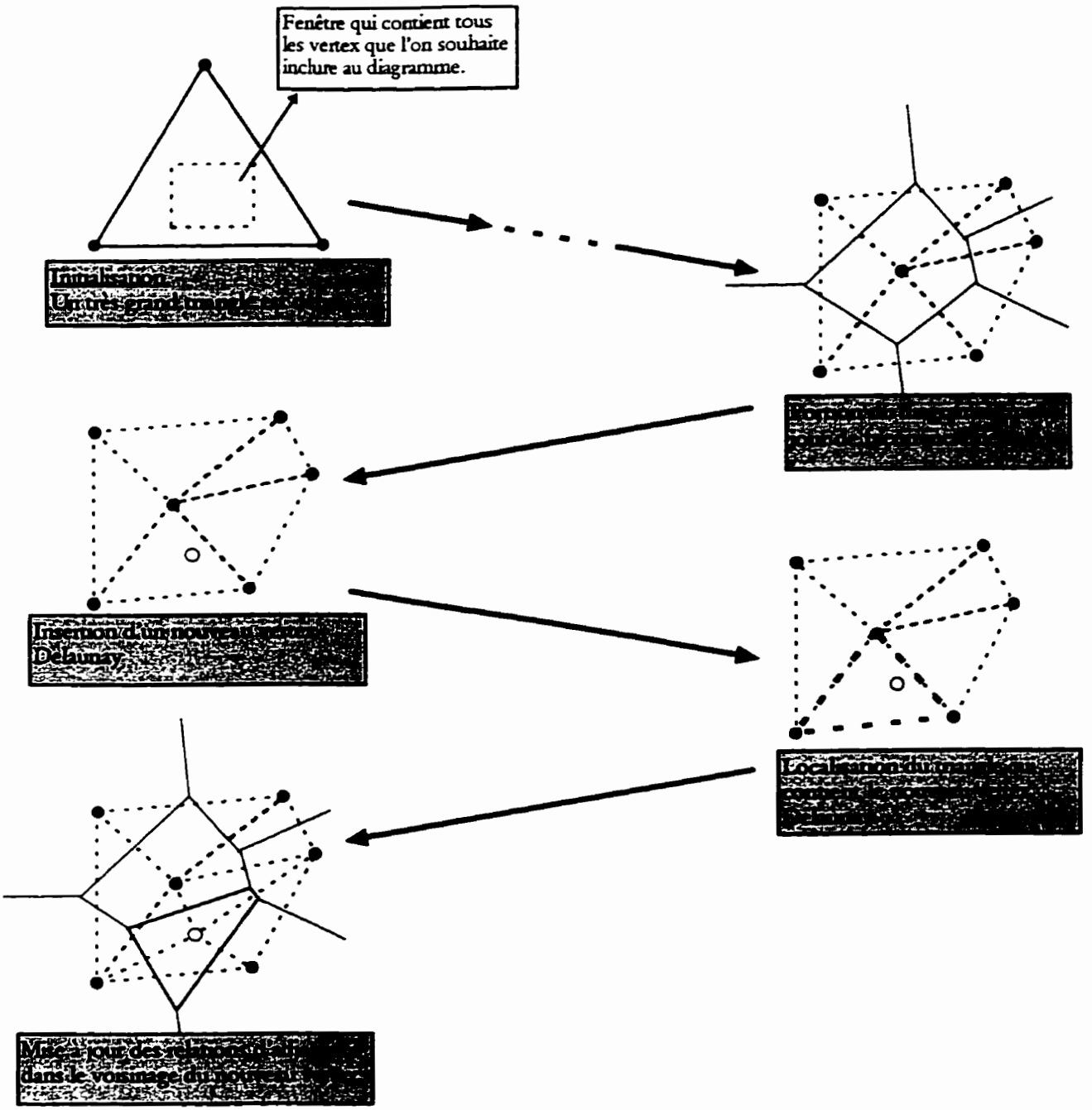

Figure 5-5 Schématisation de l'algorithme incrémentale pour la génération du diagramme Voronoï de points dans le plan.

Au regard de la figure 5.5, on constate qu'avant chaque insertion le diagramme respecte tous les critères géométriques et topologiques propres à la triangulation Delaunay et au diagramme Voronoï. Cette propriété exclusive à l'approche incrémentale permet d'en dériver une variante

pleinement dynamique permettant aussi bien l'insertion, que le déplacement et la suppression des vertex Delaunay.

5.3 Le diagramme Voronoï généralisé dans le plan.

Revenant à notre exemple des foyers d'incendie, supposons maintenant que les membres de l'équipe mettent à feu simultanément un point et une droite dans le champ de broussailles. Ils observent alors que le patron d'extinction du front incendiaire n'adopte pas le profil rectiligne observé lors de l'expérience précédente, mais plutôt une forme parabolique. Bien que surprenante au premier abord, cette observation vérifie la définition géométrique de la parabole comme étant le lieu équidistant d'une droite et d'un point.

Dans une autre expérience, ils mettent à feu simultanément deux segments de droite distincts. Ils observent à nouveaux un patron d'extinction linéaire, ce qui concorde une fois de plus avec la définition géométrique que l'on donne à une bissectrice : la frontière Voronoï entre ces deux segments recoupe bien l'angle formé entre les segments voisins en deux angles égaux. La figure 5.6 illustre ces deux derniers scénarios.

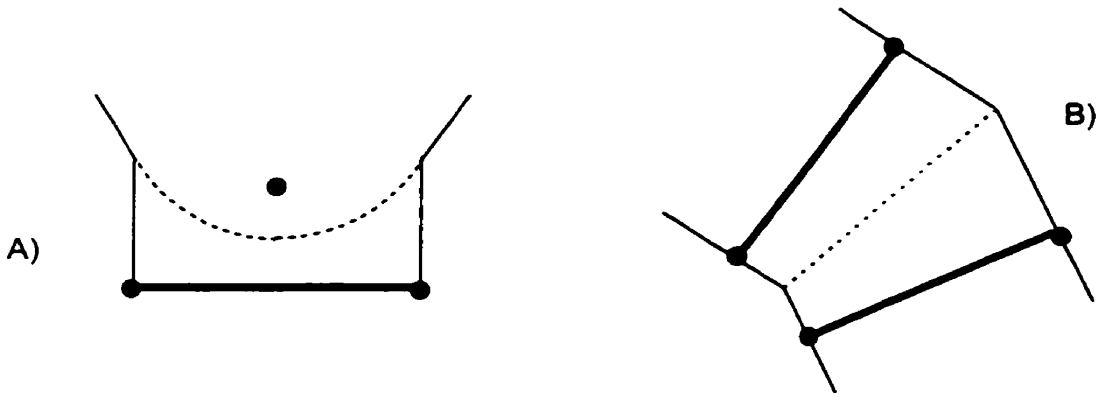

Figure 5-6 Diagramme Voronoï généralisé. En traits pointillés A) frontière parabolique entre un vertex et un segment de droite ; B) frontière linéaire entre deux segments de droite

Avec un peu plus d'efforts, l'algorithme incrémentale présenté plus tôt peut être modifié de manière à produire des diagrammes Voronoï pour des générateurs non seulement ponctuels,

mais aussi linéaires et par extension polygonaux[1]. Il est en effet possible de définir un segment de droite en terme du locus sur lequel se déplace un vertex Delaunay, un peu à la manière d'une trajectoire (Gold, 1992). D'un point de vue topologique le segment Delaunay est alors représenté comme une entité composite dont la course entre l'origine et la destination est distincte de ses extrémités (Imai, 1996). Notons qu'un segment Delaunay partage des demi-plans différents et constitue implicitement une frontière Voronoï.

Les frontières Voronoï entre les parties d'une entité composite ne sont pas considérées dans la vaste majorité des applications où ce modèle est exploité, ne préservant que les frontières séparant les entités distinctes du diagramme (Imai, 1996). Cependant, dans le cadre d'une simulation numérique, tous les lieux géométriques du domaine sont caractérisés par un ensemble de paramètres physiques qui leur sont propres, nécessitant la présence de toutes les frontières. Cette structure particulière où les frontières entre les primitives de la même entité sont conservées est appelée le diagramme Voronoï séparé (Imai, 1996). Les deux prochaines figures mettent en évidence les différences qui existent entre ces deux formes du diagramme généralisé.

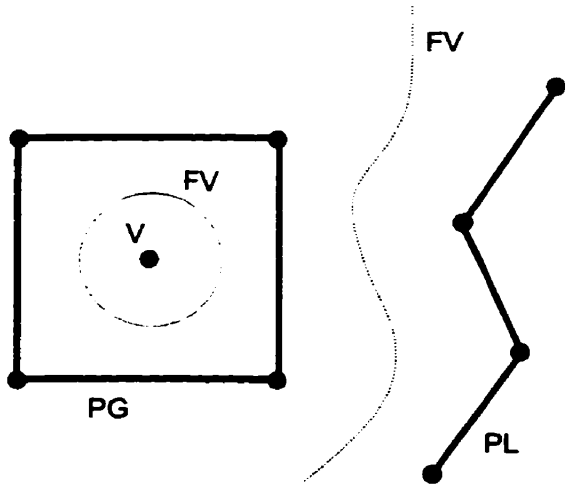

Figure 5-7 Diagramme Voronoï généralisé pour un vertex (V), une polyligne (PL) et un polygone (PG). Les frontières Voronoï sont en traits pointillés.

---

[1] Dans la pratique on considère en effet une polyligne comme une chaîne ouverte de segments jointifs et un polygone comme une chaîne fermée (Imai, 1996 ; Gold et al. 1994).

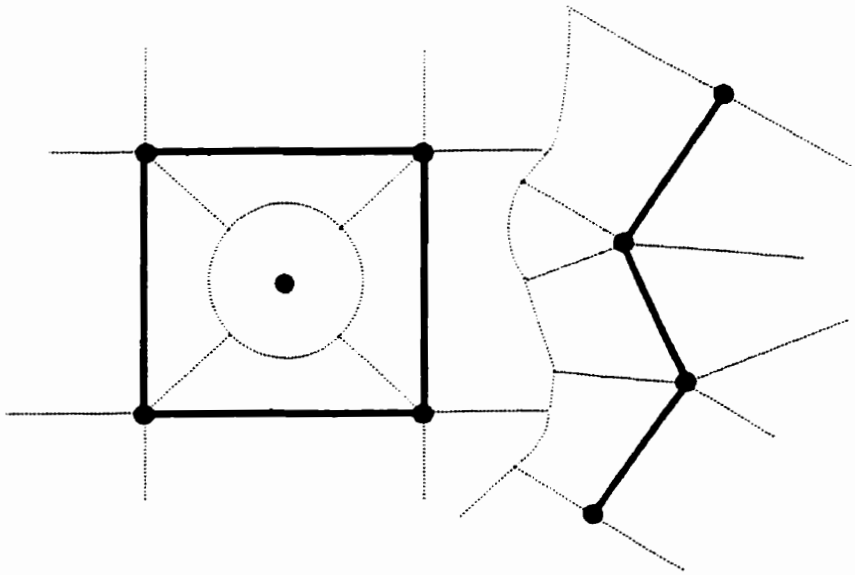

Figure 5-8 Diagramme Voronoï généralisé séparé pour un vertex, une polyligne et un polygone.

Ce sont les diagrammes Voronoï de ce dernier type, les diagrammes généralisés séparés tel que celui de la figure 5.8, que nous chercherons à exploiter comme modèle géométrique pour le support de simulations hydrogéologiques.

6 Généralisation de la IFDM.

Les primitives géométriques du modèle de données spatiales Voronoï développé par Christopher Gold permettent de représenter explicitement les lacs, rivières, lignes de crêtes, etc. d'une simulation hydrogéologique comme des entités distinctes dans une base de données géographiques. Dans ce chapitre nous tenterons de découvrir de quelle façon les diagrammes Voronoï séparés dans le plan peuvent servir à construire la géométrie d'un domaine d'écoulement et comment en dériver le système d'équations linéaires correspondant.

Ce travail tente d'établir les fondements à partir desquels un environnement SIRS interactif intégrant les simulations numériques hydrogéologiques pourrait être développé. La construction de nouvelles relations fonctionnelles entre le modèle de données spatiales Voronoï et le modèle mathématique sera mise en évidence cependant que les aspects relatifs à la résolution numérique, relevant des mathématiques appliquées plutôt que de la géomatique, seront laissés de côté.

Jusqu'à présent, le modèle géométrique sur lequel la IFDM peut être appliquée est constitué d'une tessellation irrégulière de points dont la subdivision duale minimise l'angle maximal des facettes triangulaires (Thomas, 1973). Le diagramme Voronoï dans le plan optimise cette configuration et constitue donc une structure géométrique de choix pour la mise en forme de simulations hydrogéologiques destinées à être résolues par IFDM. L'intégration de l'équation d'écoulement sur toutes les frontières d'un polygone Voronoï *'P'* quantifie la somme des flux passant par chacune des faces qui le sépare de ses *'n'* voisins *'i'*. Le terme de gauche prend alors une forme discrète unique, qui du fait demeure inchangée au regard de celle décrite par Tyson et Weber (1963) et Thomas (1973).

$$\sum_{i=1}^{i=n} T_{iP} \frac{(h_i - h_P)}{X_{iP}} \times B_{iP} = A_P S_P \frac{dh_P}{dt} - A_P R_P + A_P L_P$$

Éq. 15

L'adaptation de la IFDM au modèle de l'espace Voronoï généralisé conduit à de nouvelles formes discrètes de cette équation. Son intégration le long de frontières Voronoï séparant les primitives géométriques du modèle spatial de Christopher Gold permettra d'élargir le cadre formel proposé par Narasimhan et Witherspoon pour l'application des différences finies intégrées à la simulation des écoulements d'eau souterraine. Un soin particulier sera apporté à la description du segment Delaunay puisqu'il constitue l'entité atomique de laquelle sont construits par concaténation polylignes et polygones.

D'un point de vue purement géométrique, un segment de droite est décrit comme une succession continue de points le long d'une course linéaire entre deux extrémités. Or, dans la grande majorité des modèles hydrogéologiques usuels, l'absence de support des segments de droite dans la définition du modèle géométrique force les usagers à en construire des approximations à l'aide d'alignements très serrés de points. La figure 6.1 donne une image de ce à quoi peut ressembler l'approximation d'un segment de droite dans un diagramme Voronoï.

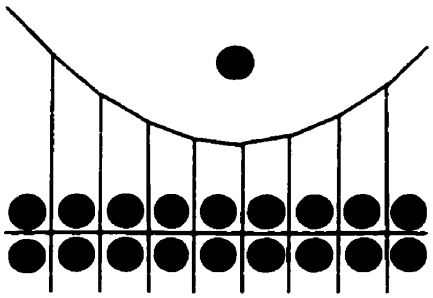

Figure 6-1 Tessellation où un segment Delaunay, opposé à un vertex, est approximé par une succession de vertex.

Dans un tel contexte, l'édition du modèle géométrique destiné à symboliser un bassin versant - avec ses cours d'eau, sites de pompages, etc. - peut s'avérer être une tâche fastidieuse, spécialement lorsqu'il n'existe aucun support à l'édition simultanée de collections de points. La représentation implicite des entités linéaires accroît aussi de façon significative la taille du système d'équations à résoudre, rendant l'exécution de la simulation gourmande en ressources et en temps.

Malgré ces faiblesses évidentes, il n'existe à ce jour aucune alternative à la représentation implicite des segments de droite dans la constitution des modèles géométriques de l'hydrogéologie. L'utilisation explicite de segments en remplacement des collections de points disjoints pourrait faciliter les tâches de création et d'édition du modèle, tout en évitant de surcharger le système d'équations linéaires. Nous proposons ici l'utilisation du modèle de l'espace Voronoï mis en forme par Christopher Gold comme substitut au modèle géométrique de la IFDM.

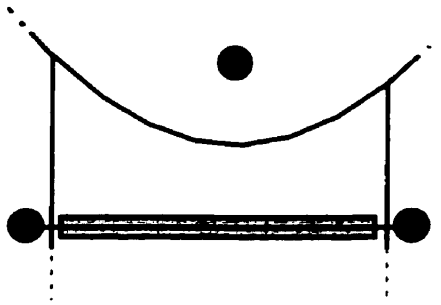

Figure 6-2 Portion d'un modèle où un segment Delaunay est construit explicitement dans le cadre du modèle de l'espace Voronoï.

Partant de la forme bidimensionnelle de l'équation d'écoulement, nous dériverons les formes discrètes permettant d'évaluer les flux passant perpendiculairement aux frontières dans un modèle discrétisé à l'aide du diagramme Voronoï séparé. Le terme de gauche de l'équation 16 (deMarsily, 1986) constitue à cet égard une articulation centrale entre les modèles mathématique et géométrique, quantifiant les écoulements d'eau dans le plan horizontal entre cellules voisines par delà les frontières qui les séparent.

$$\nabla \cdot \mathrm{T} \nabla h = S \frac{\partial h}{\partial t} - R$$

Éq. 16

Nous nous intéresserons à l'intégration de cette dernière équation le long des frontières du diagramme Voronoï séparé, cherchant à quantifier les écoulements suivants :

- entre les demi-plans bordés par un segment Delaunay par delà la frontière Voronoï implicite que représente le segment ;

- entre un segment Delaunay et ses extrémités par delà les frontières Voronoï de la forme séparée du diagramme Voronoï généralisé ;
- entre un segment Delaunay et un point par delà la frontière parabolique qui les sépare ;
- entre deux segments Delaunay par delà la frontière linéaire qui les sépare.

De l'examen de ces cas de figure seront dérivées les formes discrètes que devront prendre les termes du membre de gauche pour l'évaluation de l'équation 16 par différences finies intégrées. Ces expressions représentent ensemble l'originalité de cette recherche et viennent généraliser la IFDM en élargissant aux points, droites et polygones les entités constituantes des modèles géométriques qu'elle supporte.

6.1 Les demi-plans bornés par le segment Delaunay.

Nous verrons ici comment projeter l'équation d'écoulement sur un segment Delaunay sachant qu'il correspond aussi à une frontière Voronoï implicite séparant des demi-plans par une distance virtuellement nulle. La figure 6.3 expose plus en détail la structure intime du segment Delaunay.

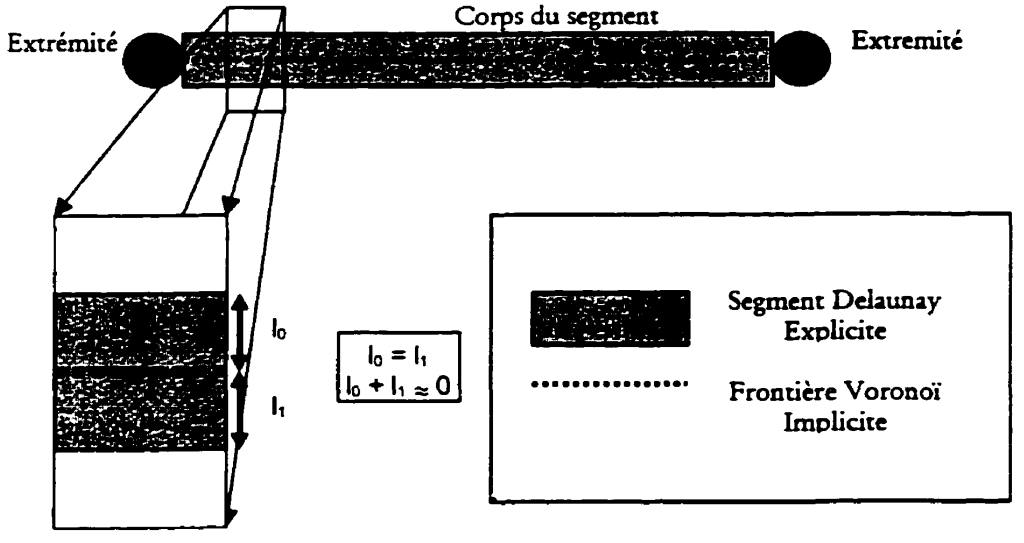

Figure 6-3 Vue rapprochée de la structure du segment Delaunay.

Afin de quantifier les phénomènes hydrogéologiques pouvant survenir le long d'un segment Delaunay, nous proposons l'utilisation de la distribution des paramètres physiques et géométriques présentée sur la figure 6.4.

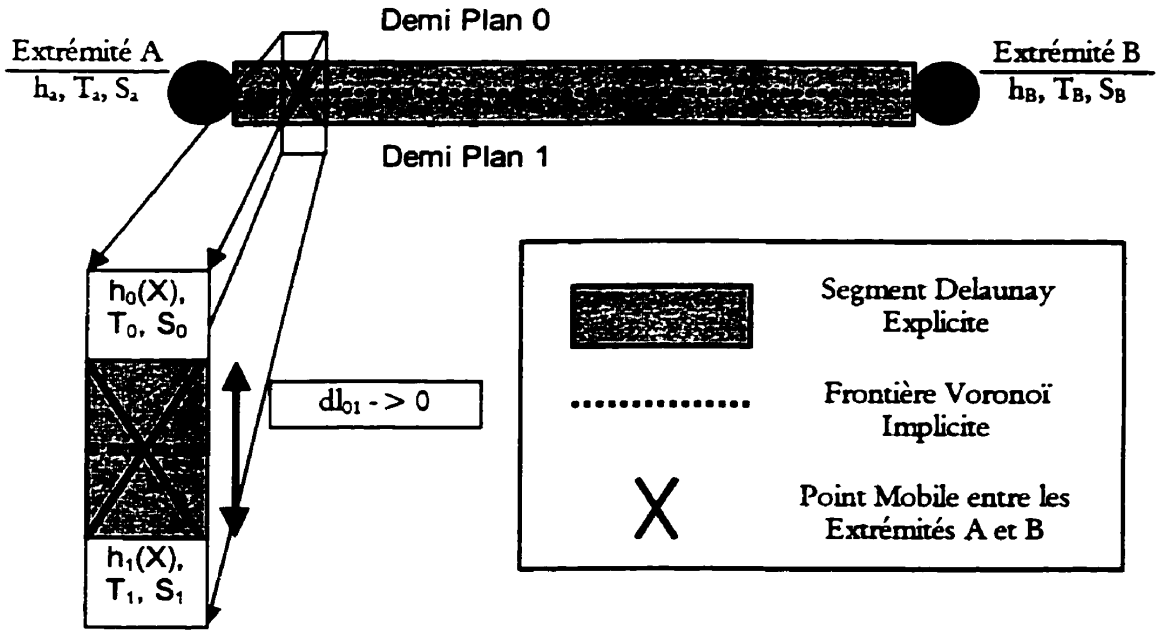

Figure 6-4 La distribution des principaux paramètres physiques et géométriques sur le segment Delaunay.

On observe, toujours sur la sur la figure 6.4, qu'il existe possiblement un écoulement d'eau perpendiculaire au segment, entre les demi-plans 0 et 1. Nous référant à ce schéma :

- Soit *'X'* un point mobile sur le segment se déplaçant entre les extrémités A et B.
- Soit *'$h_0(X)$'* la charge du demi-plan 0, fonction de la position de *'X'* sur le segment.
- Soit *'$h_1(X)$'* la charge du demi-plan 1, elle aussi une fonction de la position de *'X'*.
- Soit *'T_{01}'* la transmissivité hydraulique perpendiculaire au segment.
- Soit *'dl_{01}'* la distance séparant les demi-plans 0 et 1.

Exprimée par différences finies intégrées, l'équation 16 prend alors la forme suivante pour la frontière Voronoï implicite au segment Delaunay.

$$\int_{\Gamma}(\mathbf{T}\nabla h\cdot\mathbf{n})d\Gamma=\frac{T_{01}}{dl_{01}}\int_{A}^{B}\left[h_0(X)-h_1(X)\right]dX$$

$$\int_{\Gamma}(\mathbf{T}\nabla h\cdot\mathbf{n})d\Gamma=\lim_{dl_{01}\to 0}\left\{\frac{T_{01}}{dl_{01}}\int_{A}^{B}[h_0(X)-h_1(X)]dX\right\}=\infty$$

$$\forall h_0(X)\neq h_1(X)$$

Ce résultat partiel met tout de suite en relief une contrainte importante du modèle au niveau de la distribution des paramètres physiques pour une simulation hydrogéologique. Cette contrainte s'énonce comme suit :

La charge des demi-plans séparés par un segment Delaunay doit être la même pour toutes les positions de '*X*' sur le segment.

Toute simulation qui n'observe pas cette contrainte risque d'être numériquement instable, se traduisant par le transfert de toute l'eau qui se trouve d'un côté du segment vers l'autre côté, puis encore, laissant le système osciller indéfiniment. Mathématiquement, cette contrainte semble indiquer que le gradient hydraulique perpendiculaire à tout segment doit être nul, ce qui limiterait l'utilisation des segments Delaunay aux situations de frontières à charge constante. Il n'en est rien cependant puisque ce résultat provient plutôt du fait qu'il n'existe aucune mesure associée à 'dl_{01}'. On peut avoir a priori une connaissance du gradient perpendiculaire au segment et ainsi donc utiliser les segments Delaunay comme frontières à débit fixe.

6.2 Le segment Delaunay et ses extrémités.

Un second résultat partiel peut être dérivé de la figure 6.4, celui-là relatif à la continuité de la fonction '*h(X)*' au voisinage des extrémités A et B du segment. Nous savons qu'un gradient piézométrique non nul induit un débit hydraulique, et que pour calculer ce débit nous devons avoir une connaissance suffisante du gradient qui le module. La figure qui suit montre de quelle façon cette valeur peut être estimée.

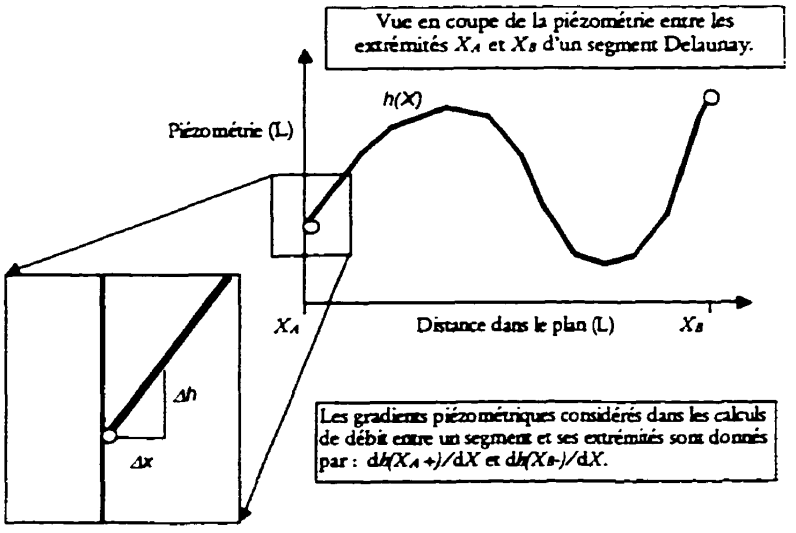

Figure 6-5 Vue en coupe de la piézométrie sur un segment Delaunay et des gradients considérés lors du calcul du débit entre le segment et ses extrémités.

En pratique, et spécialement si l'on utilise une fonction d'interpolation sur le segment Delaunay, il est très simple d'évaluer la première dérivée de la fonction piézométrique sur le corps du segment. Cependant, le montage de la figure 6.6 met en évidence une autre contrainte que doit observer toute simulation par IFDM s'appuyant sur ce modèle.

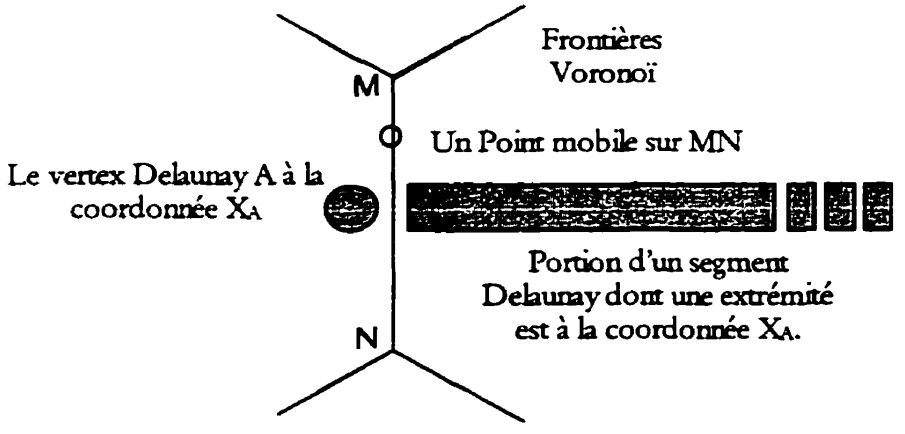

Figure 6-6 Scénario utilisé pour le calcul du débit entre un segment et une de ses extrémités.

Comme le gradient entre le segment à la coordonnée *'Xa'* et le vertex Delaunay A correspond à la dérivée première de la fonction piézométrique du segment à la coordonnée *'X_A+'*, la piézométrie *'h_A'* du vertex Delaunay A et celle du segment *'$h(X_A)$'* doivent être rigoureusement les mêmes, sans quoi l'évaluation du gradient ne tient plus étant donnée la discontinuité en C0. Il s'agit de la seconde contrainte qu'impose le système que nous tentons

d'articuler. Elle s'énonce comme suit pour un segment S dont une des extrémités coïncide à la position du vertex A :

$$h_A = h_S(X_A)$$

Toute simulation par IFDM sur un diagramme Voronoï séparé doit être assujettie à un mécanisme de contrôle garantissant la vérification de cette contrainte partout dans le domaine d'écoulement, sans quoi l'approximation des gradients aux extrémités est invalidée et le résultat des calculs qui découlent de cette hypothèse perdent leur sens.

6.3 Le segment Delaunay voisiné par un point

Les balises énoncées plus haut étant désormais en place, nous cherchons maintenant à évaluer la forme discrète du membre de gauche de l'équation 16 pour une frontière Voronoï de forme parabolique séparant un point et un segment Delaunay. La figure 6.7 présente le cas général qui servira à construire l'expression recherchée.

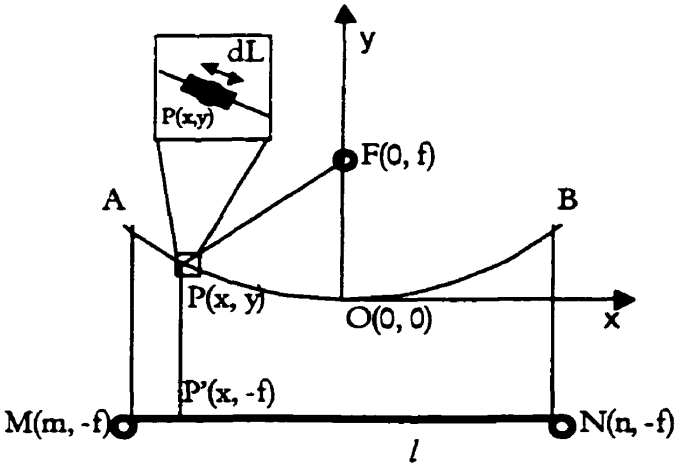

Figure 6-7 Cas général où le segment *l* parallèle à l'axe x et liant M à N est le voisin d'un vertex F dans un diagramme Voronoï généralisé, le foyer de la parabole passant par A, P et B.

Suivant l'approche suggérée par Narasimhan et Witherspoon (1976), nous trouverons la forme discrète de l'équation gouvernante en intégrant sur la frontière parabolique AB les débits prenant place entre F et chacun des points de *l*. Ainsi, pour chaque position P'(x, -f) sur la

droite, étant la projection perpendiculaire du point P(x, y) de la parabole, nous avons un débit infinitésimal dont la grandeur est donnée par :

$$dq = T_{FP'(x,-f)} \frac{\left(h_{P'(x,-f)} - h_F\right)}{X_{FP'(x,-f)}} dL$$

Éq. 17

L'équation 17 introduit quelques nouveaux termes :

- la partie infinitésimale *dL* exprime une petite longueur d'arc sur la parabole,
- la piézométrie $h_{P'}$ sur *l* varie avec la position de P' sur MN,
- la distance $X_{FP'}$ séparant F de P' varie avec la position de P' sur MN,
- la transmissivité hydraulique $T_{FP'}$ est aussi une fonction de la position de P' sur MN.

Pour résoudre cette équation, nous procédons à quelques changements de variable de manière à exprimer dL, $h_{P'}$, $X_{FP'}$ et $T_{FP'}$ selon la position de P' sur MN. Posant comme contrainte que la transmissivité hydraulique demeure constante le long d'un segment, nous pouvons isoler le terme $T_{FP'}$ de l'influence de P', simplifiant quelque peu le problème à résoudre[2]. Pour l'évaluation des termes restants, la fonction de la parabole séparant F de *l* constitue un élément essentiel à la solution. Soit :

$$x^2 = 4fy$$

$$y = \frac{x^2}{4f}$$

Éq. 18

Posant qu'une longueur d'arc infinitésimale dL sur une fonction polynomiale en x est donnée par :

---

[2] Notons que cette décision est tout à fait arbitraire, étant d'une part basée sur le fait qu'il n'existe pas de véritable consensus relatif à la « meilleure » façon d'interpoler la transmissivité hydraulique, et d'autre part sur le fait que le but de ce travail est d'explorer une nouvelle approche plutôt que de proposer une solution complète.

$$dL = \sqrt{dx^2 + dy^2}$$

$$dL = \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx$$

on obtient :

$$dL = \sqrt{1 + \frac{x^2}{4f^2}}dx$$

Éq. 19

Maintenant, pour tout point P de la parabole donnée par l'équation 18, la fonction $X_{FP'}$ évaluant la distance entre F et P' est donnée par :

$$X_{XP'} = \sqrt{x^2 + 4f^2}$$

Éq. 20

Pour ce qui est de la piézométrie $h_{P'}$, nous devons choisir une fonction d'interpolation sur le segment MN qui tienne compte des facteurs suivants :

- des contraintes fixes existent aux extrémités du segment ;
- un surplus ou un déficit d'eau peut s'accumuler dans la cellule associée à MN ;
- les paramètres de la fonction d'interpolation sont autant de nouvelles inconnues à résoudre lors de la simulation numérique.

Sans rechercher un haut degré de sophistication, nous choisissons ici une fonction polynomiale du second degré. Cette fonction respecte les contraintes énumérées ci-haut et minimise le nombre de paramètres introduits dans le système d'équation. D'autres fonctions d'interpolation pourraient facilement se substituer à celle-ci dans la mesure ou elles tiennent compte des facteurs énumérés plus haut. Cette expression est associée au segment, ce qui permet de définir une quantité pour l'objet lui-même sans autre support. Ainsi :

$$h_{P'} = \alpha\lambda^2 + \beta\lambda + \chi$$

Éq. 21

où λ varie de 0 à 1 entre les extrémités du segment. Pour un segment borné par M et N nous pouvons transformer l'équation 21 en :

$$h_{P'} = \alpha[\lambda(x)]^2 + \beta\lambda(x) + \chi$$
$$h_{P'} = \alpha\left(\frac{x-m}{n-m}\right)^2 + \beta\left(\frac{x-m}{n-m}\right) + \chi$$

Éq. 22

Finalement, reprenant l'équation 17 et y substituant les expressions 19, 20 et 22 pour une valeur de $T_{FP'}$ constante T_{Fl}, on obtient l'équation du débit instantané entre un segment et un point, soit :

$$dq = T_{Fl}\frac{\left(\alpha[\lambda(x)]^2 + \beta\lambda(x) + \chi - h_F\right)}{\sqrt{x^2+4f^2}}\sqrt{1+\frac{x^2}{4f^2}}dx$$
$$dq = T_{Fl}\frac{\left(\alpha[\lambda(x)]^2 + \beta\lambda(x) + \chi - h_F\right)}{\sqrt{x^2+4f^2}}\sqrt{\frac{x^2+4f^2}{4f^2}}dx$$
$$dq = \frac{T_{Fl}}{2f}\left(\alpha[\lambda(x)]^2 + \beta\lambda(x) + \chi - h_F\right)dx$$

L'intégration de cette équation entre M et N permet d'évaluer le débit qui prend place entre le segment MN et le vertex F. Nous obtenons :

$$Q_{Fl} = \int dq$$

$$Q_{Fl} = \int_m^n \frac{T_{Fl}}{2f}\left(\alpha[\lambda(x)]^2 + \beta\lambda(x) + \chi - h_F\right)dx$$

$$Q_{Fl} = \frac{T_{Fl}}{2f}\left[\alpha\int_m^n [\lambda(x)]^2 dx + \beta\int_m^n \lambda(x)dx + (\chi - h_F)\int_m^n dx\right]$$

$$Q_{Fl} = \frac{T_{Fl}}{2f}\left[\frac{\alpha}{(n-m)^2}\int_m^n (x^2 - 2xm + m^2)dx + \frac{\beta}{(n-m)}\int_m^n (x-m)dx + (\chi - h_F)\int_m^n dx\right]$$

$$Q_{Fl} = \frac{T_{Fl}}{2f}\left[\frac{\alpha}{(n-m)^2}\left(\frac{x^3}{3} - mx^2 + m^2x\right) + \frac{\beta}{(n-m)}\left(\frac{x^2}{2} - xm\right) + (\chi - h_F)x\right]_m^n$$

Éq. 23

Cette dernière équation quantifie le débit entre un vertex et un segment Delaunay séparés par une frontière parabolique dans une simulation par IFDM utilisant le diagramme Voronoï séparé dans le plan comme mode de discrétisation. Il s'agit d'une solution générique qui peut être adaptée aux situations où un demi-plan est voisiné par plus d'une entité ponctuelle par un simple changement des bornes d'intégration. De la même manière, les segments Delaunay colinéaires partageant une extrémité sont fondus en un seul, levant l'indétermination relative au gradient effectif au point de jonction provoquée par la discontinuité en C1 de la piézométrie au lieu de l'extrémité partagée.

6.4 Le segment Delaunay voisiné par un second segment Delaunay.

Suivant une approche similaire, nous cherchons maintenant à exprimer sous une forme discrète le membre de gauche de l'équation 16 pour une frontière Voronoï séparant une paire de segments Delaunay voisins. La figure 6.8 présente le cas général à partir duquel l'équation sera dérivée :

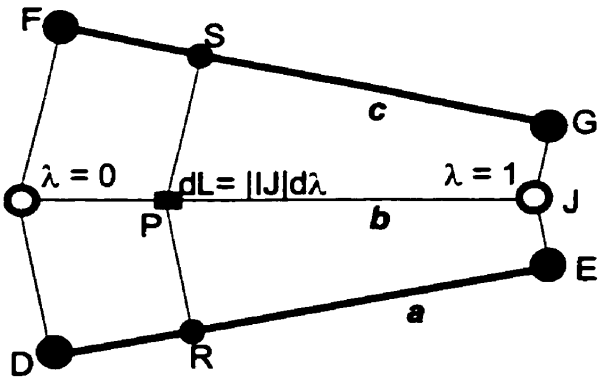

Figure 6-8 Scénario illustrant la position d'un point mobile P sur l'interface b séparant les domaines Voronoï associés aux segments a et c.

Sans nier l'existence d'autres combinaisons, supposons ici le cas où le paramètre λ varie sur a de 0 à 1 entre D et E, et sur b entre F et G. Le débit instantané entre c et a est donné par :

$$dq = T_{RS}\frac{(h_R - h_S)}{|RS|}dL$$
$$dq = T_{ac}\frac{(h_R - h_S)}{|RS|}dL$$

Éq. 24

Tout comme nous l'avons fait plus tôt, nous supposerons une fois de plus que la transmissivité hydraulique est constante partout entre a et c. À l'examen de la dernière figure on constate que plusieurs des quantités restantes sont linéairement dépendantes de λ. Ainsi, l'équation 24 devient :

$$dq = T_{ac}|IJ|\frac{(\alpha_a - \alpha_c)\lambda^2 + (\beta_a - \beta_c)\lambda + (\chi_a - \chi_c)}{\lambda|GE| + (1-\lambda)|FD|}d\lambda$$

Éq. 25

L'intégration de cette dernière équation pour λ entre 0 et 1 nous retourne l'expression permettant d'évaluer le débit prenant place entre les segments a et c par delà l'interface b. On obtient alors :

$$Q_{ac} = \int dQ$$

$$Q_{ac} = K_{ac}|IJ|\int_0^1 \left[\frac{(\alpha_a - \alpha_c)\lambda^2 + (\beta_a - \beta_c)\lambda + (\chi_a - \chi_c)}{\lambda|GE| + (1 - \lambda)|FD|}\right]d\lambda$$

$$Q_{ac} = K_{ac}|IJ|\left[\begin{array}{l}(\alpha_a - \alpha_c)\int_0^1 \frac{\lambda^2 d\lambda}{\lambda(|GE| - |FD|) + |FD|} + \\ (\beta_a - \beta_c)\int_0^1 \frac{\lambda d\lambda}{\lambda(|GE| - |FD|) + |FD|} + \\ (\chi_a - \chi_c)\int_0^1 \frac{d\lambda}{\lambda(|GE| - |FD|) + |FD|}\end{array}\right]$$

Cette équation est résolue en :

Soit:

$$a = |FD|$$

$$b = |GE| - |FD|$$

$$Q_{ac} = K_{ac}|IJ|\left\{\begin{array}{l}\frac{(\alpha_a - \alpha_c)}{2b^3} * [(a + b\lambda) - 4a(a + b\lambda) + 2a^2 In|a + b\lambda|] + \\ \frac{(\beta_a - \beta_c)}{b^2} * [(a + b\lambda) - aIn|a + b\lambda|] + \\ \frac{(\chi_a - \chi_c)}{b} * (In|a + b\lambda|)\end{array}\right\}\Bigg|_{\lambda=0}^{\lambda=1}$$

Éq. 26

Bien que ce travail ne suggère aucun schème de résolution numérique, l'Annexe 1 permet d'apprécier la complexité algorithmique de chaque itération dans le but d'évaluer la valeur des paramètres α, β et χ définis sur chaque segment.

L'équation 26 permet de quantifier la quantité d'eau qui se déplace de a vers c, deux segments Delaunay voisins dans un diagramme Voronoï séparé. Dans le cas particulier ou a et c partagent une extrémité, l'équation demeure valide étant donnée la continuité en C0 de la piézométrie au lieu de cette extrémité, tel que spécifié un peu plus tôt. Dans le cas où a et c ne forment qu'un seul et même segment - i.e. lorsque a et c partagent les mêmes extrémités - le débit est nul, vérifiant du même coup la contrainte énoncée à cet égard dans la section 6.1.

6.5 Le nouveau portrait de la IFDM

Les sections 6.1 à 6.4 constituent l'essentiel de cet effort de recherche, présentant les expressions qui définissent le nouveau portrait de la IFDM pour la simulation des écoulements d'eau souterraine. Le degré d'abstraction qui caractérise cette nouvelle forme de la IFDM permet désormais l'évaluation des transferts d'eau entre entités géométriques voisines 0 et 1-dimensionnelles, nommément des points et des segments de droite dans un diagramme Voronoï généralisé séparé.

7 Une validation de la IFDM pour le diagramme Voronoï de points.

Nous proposons maintenant une réalisation de la IFDM pour les diagrammes Voronoï de points dans le plan de manière à valider l'approche pour des cas simples. Dans un premier temps, nous produirons les solutions analytiques pour un scénario synthétique connu. Nous répéterons ensuite le même exercice, mais cette fois pour les solutions numériques par FDM obtenues à l'aide du logiciel «*Aquifer Simulation Model*» (ASM). Nous présenterons finalement le système que nous avons développé par IFDM avant de comparer les uns aux autres les résultats obtenus.

Bien que la validation de l'approche ait déjà été produite par Narasimhan et Witherspoon, il demeure souhaitable de la répéter d'autant plus que ce travail formalise l'utilisation de la structure Voronoï pour des ensembles de points pour la représentation géométrique des domaines d'écoulement. Notons que la réalisation de la IFDM pour des diagrammes Voronoï généralisés n'a pu être entreprise dans le cadre de cette recherche puisque la réalisation de l'engin Voronoï développé par Christopher Gold et son équipe n'était pas complète.

7.1 Un modèle spatial Voronoï simplifié.

Ce prototype repose sur une version simplifiée du modèle spatial Voronoï en ce qu'il ne structure que des ensembles de points dans le plan. Il utilise la structure de données *QuadEdge* introduite par Guibas et Stolfi (1985). De toutes les structures de données proposées à ce jour pour la représentation des diagrammes Voronoï de points dans le plan, le *QuadEdge* constitue certainement l'une des plus populaires pour son caractère générique et sa simplicité.

En termes simples, le *QuadEdge* sert à maintenir les relations de voisinage entre les points, les segments et les polygones qui constituent une subdivision du plan. En termes plus élaborés, le *QuadEdge* sert à représenter la topologie de tout graphe inscrit dans le plan, un graphe étant une collection d'arêtes reliant une collection de points, à la manière d'un réseau. Le *QuadEdge* représente à la fois le graphe, ici un diagramme Voronoï, et son dual, ici la

triangulation Delaunay. Seuls deux opérateurs topologiques et trois évaluateurs géométriques suffisent à la construction et à la modification d'un diagramme Voronoï.

Sur toute surface plane et partout sur celle-ci, il existe deux façons de définir localement sur un disque une rotation en sens horaire. On définit l'orientation d'un élément X en associant à X l'orientation du disque qui le contient. Pour l'application particulière qui nous intéresse, une seule possibilité ne sera retenue, soit la «vue en plan».

De la même manière, il existe deux façons d'établir un ordre entre les extrémités d'un segment pour établir la direction de celui-ci. Pour tout segment *e* orienté et dirigé, il est ainsi possible de nommer sans équivoque son origine et sa destination, sa droite et sa gauche. C'est en partie sur la base de ces observations élémentaires que le *QuadEdge* a vu le jour. Ainsi, utilisant ici la syntaxe du langage C++, le *QuadEdge* est défini comme suit :

```
class Edge
{
public :
        // Accesseurs
        //--------------------
        Edge Next( void )
        {
                return mNext;
        }

        Edge Rot( void )
        {
                return( ( mRot < 3 ) ? ( this + 1 ) : ( this - 3 ) );
        }

private :
        // Associations
        //--------------------
        Data mData;         // Pointeur sur le noeud d'où e émerge.

        Edge mNext;                 // Pointeur sur le prochain segment en sens anti-
                                    // horaire émergeant du même noeud.

        int mRot;           // Indice de e dans le QuadEdge.
};
typedef Edge[ 4 ] QuadEdge;
```

Cet alphabet simple permet l'élaboration d'une algèbre sophistiquée pour la visite du voisinage immédiat de tout segment *e* dans une subdivision du plan. La figure qui suit isole une partie d'une triangulation Delaunay (en traits pleins) et présente les opérateurs de cette

algèbre. Les étiquettes attachées aux segments du diagramme correspondent aux valeurs retournées par ces opérateurs pour le segment *e*.

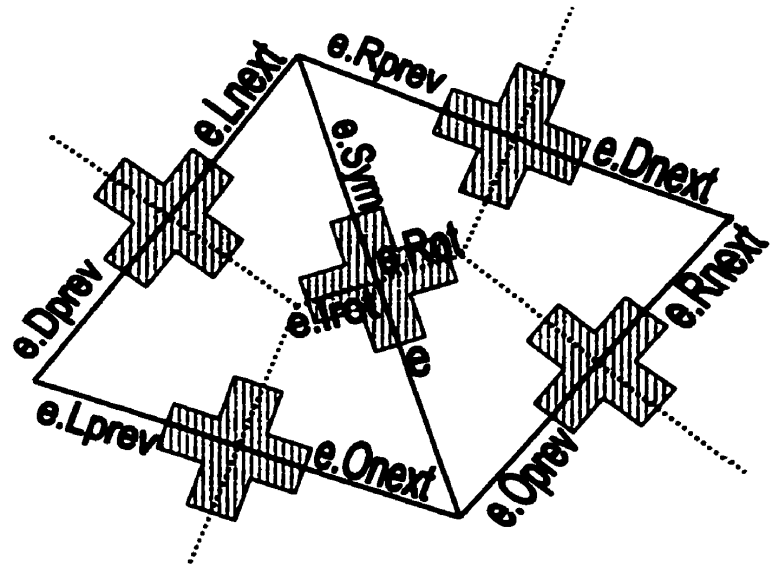

Figure 7-1 Les méthodes du *QuadEdge*

Ces méthodes permettent l'accès à la structure du graphe de proche en proche, facilitant son défilement selon un algorithme quelconque, que ce soit celui développé par Christopher Gold pour le défilement des données de digitalisation ou bien l'algorithme du chemin le plus court.

Pour construire une subdivision du plan à l'aide du *QuadEdge*, Guibas et Stolfi (1985) proposent deux opérateurs topologiques de base et complètent ce jeu de fonctions en développant des méthodes élaborées à partir des opérateurs de base. Le premier opérateur, *Edge MakeEdge()*, est en quelque sorte le constructeur de *QuadEdge*. Il retourne une structure isolée, proprement initialisée, prête à être insérée dans une subdivision. La valeur des champs mRot (indices) et mNext (pointeurs) pour chacun des segments du *QuadEdge* retourné par *MakeEdge()* sont apparentes sur la figure 7.2.

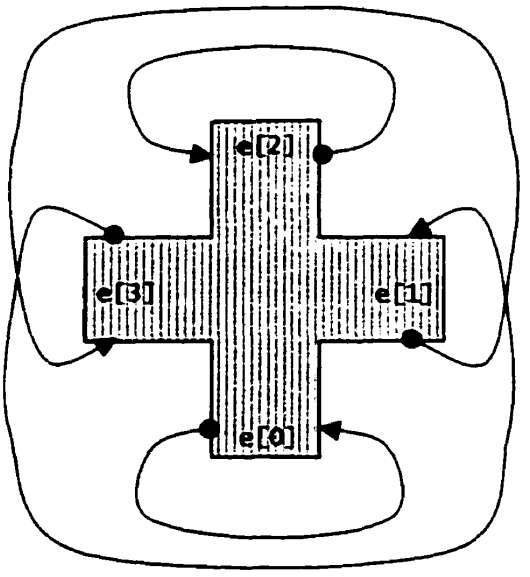

Figure 7-2 Valeurs de mRot et mNext pour un QuadEdge retourné par l'opérateur MakeEdge()

Le second opérateur se nomme *void Splice(Egde a, Edge b)*. Cet operateur est au cœur de toute opération visant l'ajout ou encore le retrait d'un *QuadEdge* à une subdivision du plan. En effet, *Splice* connecte deux segments disjoints, et inversement déconnecte deux segments jointifs. La figure 7.3 suggère les modifications aux valeurs mNext des segments impliqués dans l'opération. Ici, *a* appartient à un *QuadEdge* tout juste créé à l'aide de *MakeEdge()*.

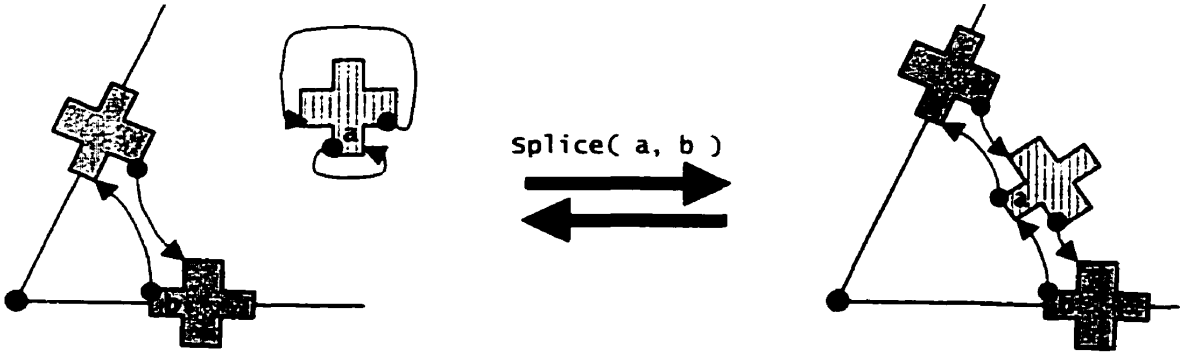

Figure 7-3 Effet de l'opérateur Splice(a, b) sur les pointeurs mNext affectés

Ainsi donc, le *QuadEdge* et ses méthodes nous permettent de construire et de représenter tout graphe planaire d'une manière consistante, à partir d'un nombre restreint d'opérateurs topologiques. Le *QuadEdge* se prête particulièrement bien à la construction et au maintient du diagramme Voronoï de points dans le plan puisqu'il s'agit d'une structure appartenant à la famille des graphes dit fortement connectés. Pour ce faire, en plus des méthodes de défilement et de modification du graphe, trois évaluateurs géométriques sont nécessaires à sa construction

comme nous l'avons déjà dit. Utilisant toujours la nomenclature développée par Guibas et Stolfi, et depuis adoptée par une grande partie de la communauté des algorithmiciens, on trouve en premier lieu les évaluateurs *bool rightOf(Edge e, Vertex v)* et *bool leftOf(Edge e, Vertex v)* qui déterminent de quel côté le point *v* est situé par rapport au segment orienté *e*. Ces deux fonctions reposent en fait sur l'évaluation de déterminant d'une matrice de 3X3, fonction dénommée *CCW*. Soit *o* l'origine de *e*, et *d* sa destination. Pour la fonction *rightOf* nous avons :

$$\begin{vmatrix} x_d & y_d & 1 \\ x_o & y_o & 1 \\ x_v & y_v & 1 \end{vmatrix} > 0$$

Éq. 27

Et pour *leftOf*, *CCW* est évaluée avec les valeurs suivantes :

$$\begin{vmatrix} x_o & y_o & 1 \\ x_d & y_d & 1 \\ x_v & y_v & 1 \end{vmatrix} > 0$$

Éq. 28

Le troisième évaluateur géométrique requis pour la construction du diagramme Voronoï de points dans le plan se nomme *bool inCircle(Vertex a, Vertex b, Vertex c, Vertex d)*. Il est utilisé pour déterminer si oui ou non le point *d* est à l'intérieur du cercle passant par les points *a*, *b* et *c*. Cette fonction prend aussi la forme d'un déterminant, mais cette fois pour une matrice de 4X4. Ainsi, nous avons :

$$\begin{vmatrix} x_a & y_a & \left(x_a^{\,2} + y_a^{\,2}\right) & 1 \\ x_b & y_b & \left(x_b^{\,2} + y_b^{\,2}\right) & 1 \\ x_c & y_c & \left(x_c^{\,2} + y_c^{\,2}\right) & 1 \\ x_d & y_d & \left(x_d^{\,2} + y_d^{\,2}\right) & 1 \end{vmatrix} > 0$$

Éq. 29

Suivant l'algorithme incrémentale introduit à la section 5.2.2, nous avons tout d'abord à localiser le triangle qui contient le point que nous tentons d'insérer au diagramme. Pour ce faire, nous devons disposer d'une fonction nous permettant de traverser le graphe à la recherche de ce triangle unique. Cette méthode, *Edge locate(Vertex v)*, retourne un des segments *e* délimitant ce triangle de telle sorte que *v* est à sa gauche. Notons que la voie que nous avons adoptée diffère quelque peu de celle proposée par Guibas et Stolfi : la première condition qui teste si *v* est à la droite du segment *e* a été extraite de la boucle et placée devant celle-ci. Comme cette condition ne peut être vérifiée qu'une fois sur deux, et ce lors de la toute première itération, cette modification constitue une optimisation qui peut faire une différence importante pour des ensembles de points de grande dimension. Le code Java fourni à l'annexe B peut être consulté pour avoir un aperçu de l'algorithme d'insertion du point une fois le triangle localisé.

```
Edge locate( Vertex v )
{
        Edge e = un segment quelconque appartenant déjà à la subdivision;
        if( rightOf( e, v ) )
        {
                e = e.sym;
        }
        while( true )
        {
                if( v == e.org )
                {
                        return e;
                }
                else if( v == e.dest )
                {
                        return e.sym;
                }
                else if( !rightOf( e.onext, v ) )
                {
                        e = e.onext;
                }
                else if( !rightOf( e.dprev, v ) )
                {
                        e = e.dprev;
                }
                else
                {
                        return e;
                }
        }
}
```

Dans le cadre particulier de la IFDM, la constitution du diagramme Voronoï :

- définit la structure géométrique supportant les simulations ;

- établit la connectivité des nœuds entre eux ;
- supporte le défilement du graphe par le biais d'une interface simple.

Le modèle spatial Voronoï appliqué à la structuration des modèles de la IFDM permet de consolider les données géométriques et descriptives en une représentation logique unique qui demeure valide à travers les phases d'édition, de simulation, d'analyse et de présentation des résultats.

7.2 Le scénario hydrogéologique synthétique.

Pour valider une nouvelle approche numérique, on doit pouvoir reproduire les résultats donnés par les équations du modèle analytique. On admet aussi avec prudence la comparaison des résultats de nouvelles méthodes avec ceux obtenus à l'aide d'une solution numérique éprouvée.

La première étape est donc ici d'identifier laquelle des équations analytiques sera utilisée ; nous devons choisir un scénario synthétique que devront reproduire les modèles numériques. Nous avons choisi ici un scénario bien connu des hydrogéologues, soit celui la nappe libre à écoulements convergents en régime permanent dont le modèle mathématique est connu sous le nom de l'équation de Dupuit-Forchheimer. On peut décrire le contexte hydrogéologique de la nappe à écoulements convergents comme une île parfaitement circulaire au milieu de laquelle se trouve une station de pompage. La piézométrie est connue au lieu du puits de même qu'à la marge de l'île, où les eaux de surface infiltrent le sol de l'île. La figure 7.4 illustre une vue en coupe de ce scénario :

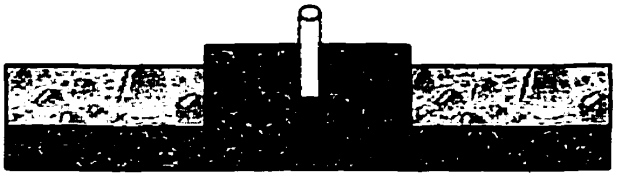

Figure 7-4 Vue en coupe du scénario de la nappe à écoulements convergents en régime permanent.

L'équation simplifie le problème en le ramenant à un problème à deux dimensions exprimé sur un plan vertical qui passe par le puits. Cette équation, pour une nappe libre en régime d'écoulement permanent, prend la forme suivante, soit :

$$h(x) = \sqrt{y^2 + \frac{(Y^2 - y^2)\ln\left(\frac{x}{r}\right)}{\ln\left(\frac{R}{r}\right)}}$$

Éq. 30

Le paramètre y est la piézométrie au centre du puits, Y est la piézométrie en marge de l'île, r est le rayon du puits et R est le rayon de l'île. Cette équation permet d'évaluer le profil de la surface piézométrique en plaçant le centre du puits à l'origine.

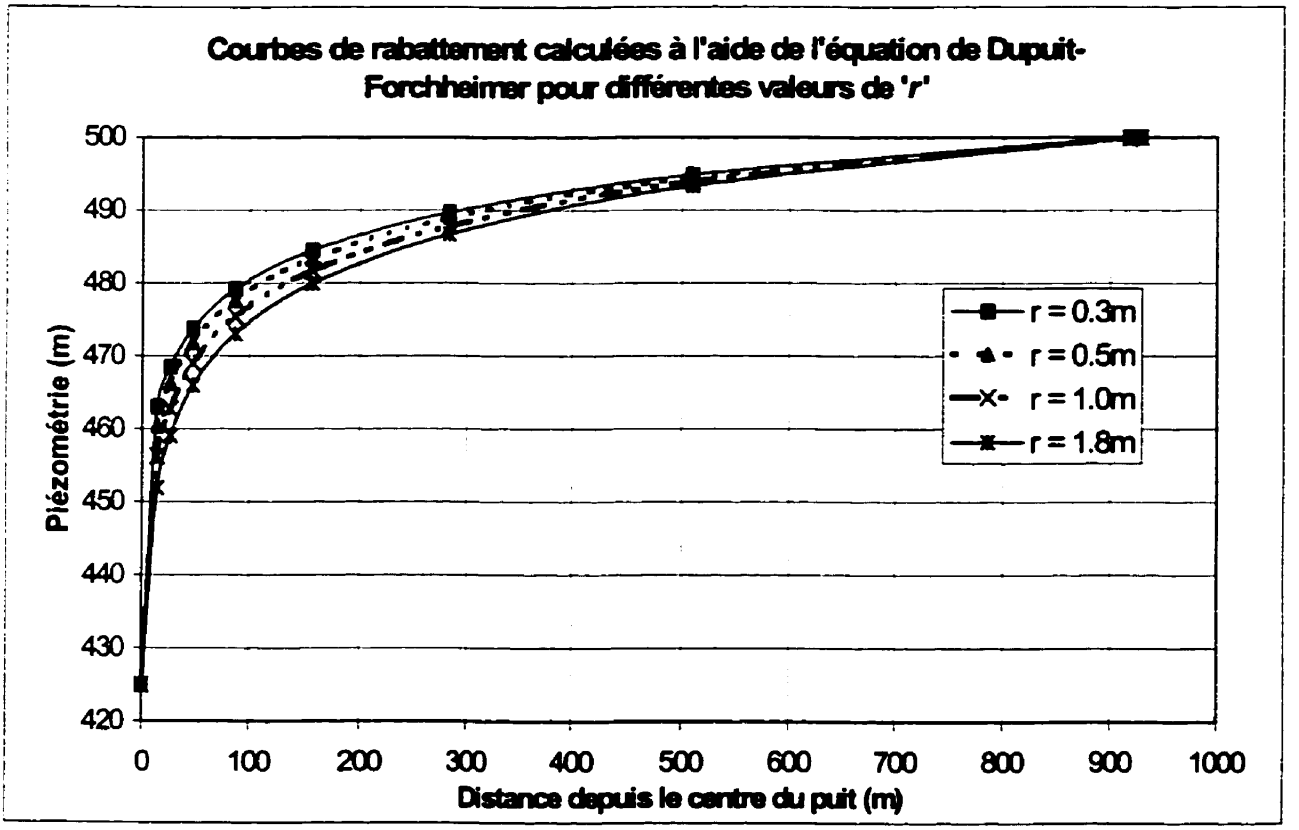

Figure 7-5 Courbes de rabattement pour y = 425m, Y = 500m, R = 928m et différentes valeurs de r.

Le graphique de la figure 7.5 montre quelques courbes de rabattement obtenues en faisant varier le rayon du puits r pour les mêmes valeurs de y, Y et R. Comme on peut l'observer, plus

le rayon du puits est grand, plus l'influence du pompage se fait sentir à distance. Ces courbes et les paramètres utilisés pour les produire sont désormais considérées comme la référence pour la suite de cette validation.

7.3 Une solution numérique validée.

La seconde solution que nous produisons ici pour valider la IFDM pour les nappes à écoulements convergents en est une numérique. Nous ferons usage du logiciel commercial ASM, mis au point par Kinzelbach (1989). Il s'agit d'une version en QuickBasic de la méthode conventionnelle des différences finies, la FDM.

Pour reproduire le scénario de la nappe à écoulements convergents présenté à la section 7.2, nous utilisons une grille régulière de 13X13 tuiles, chacune ayant 154.7m de côté. Les eaux de surface entourant l'île ont été modélisées comme étant des cellules à charge constante. Il en a été de même pour le puits de pompage. La figure 7.6 montre le modèle par lequel la simulation par FDM a été conduite. Modéliser une masse circulaire avec des tuiles orthogonales n'est pas une mince affaire....

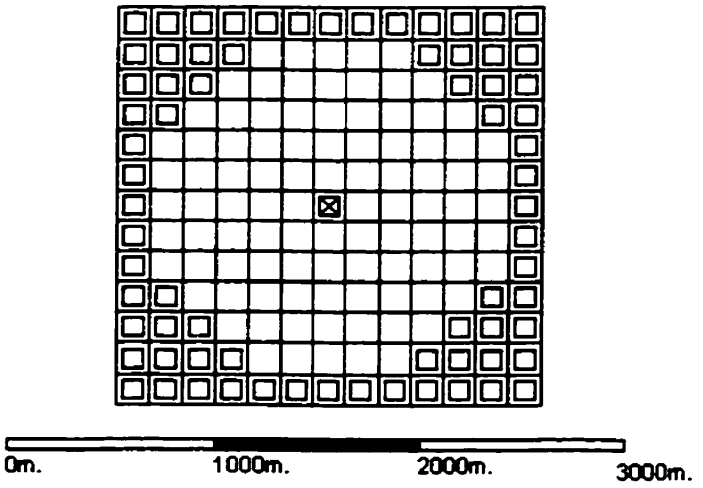

Figure 7-6 Modèle géométrique approximant le scénario de la nappe à écoulements convergents.

Les tuiles marquées d'un carré blanc sont les tuiles ou la piézométrie est considérée comme constante. Le puits, au centre du domaine, est caractérisé par une piézométrie de 425m.,

pendant que les tuiles de la périphérie ont une piézométrie de 500m. Le rayon est approximativement de 928m. Les figures 7.7 et 7.8 donnent les résultats de la simulation.

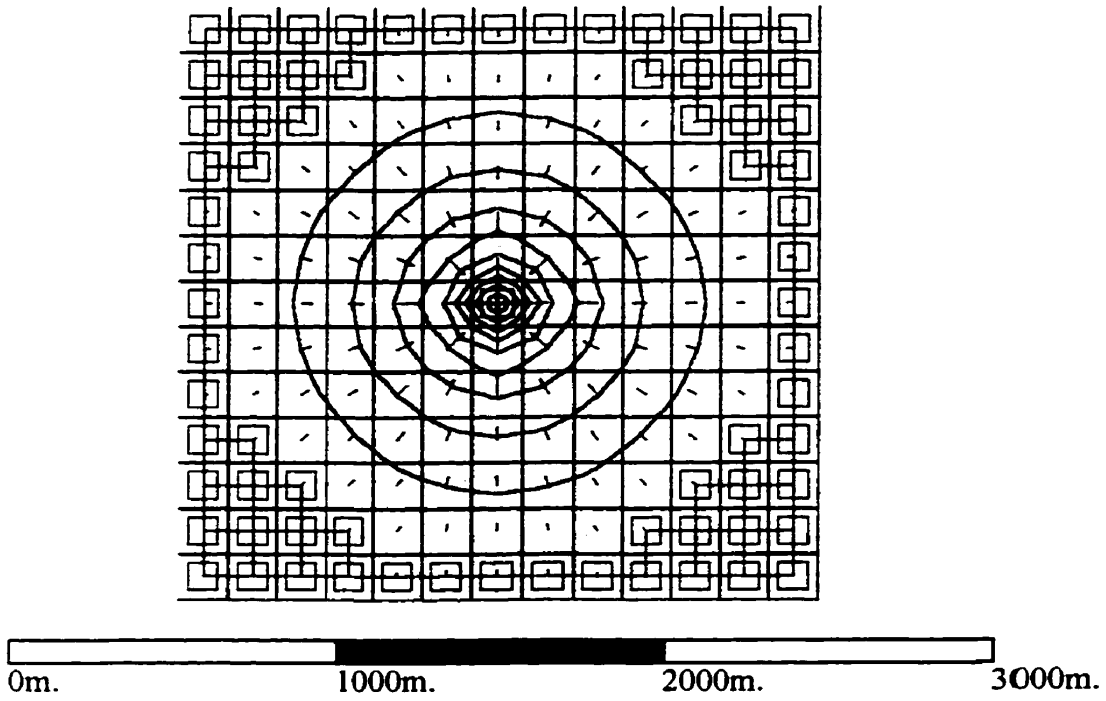

Figure 7-7 Vue en plan des résultats de la simulation par différences finies produite par ASM. Les courbes concentriques représentent les courbes de nivaux de la surface piézométrique, distantes de 7,5m selon l'axe vertical.

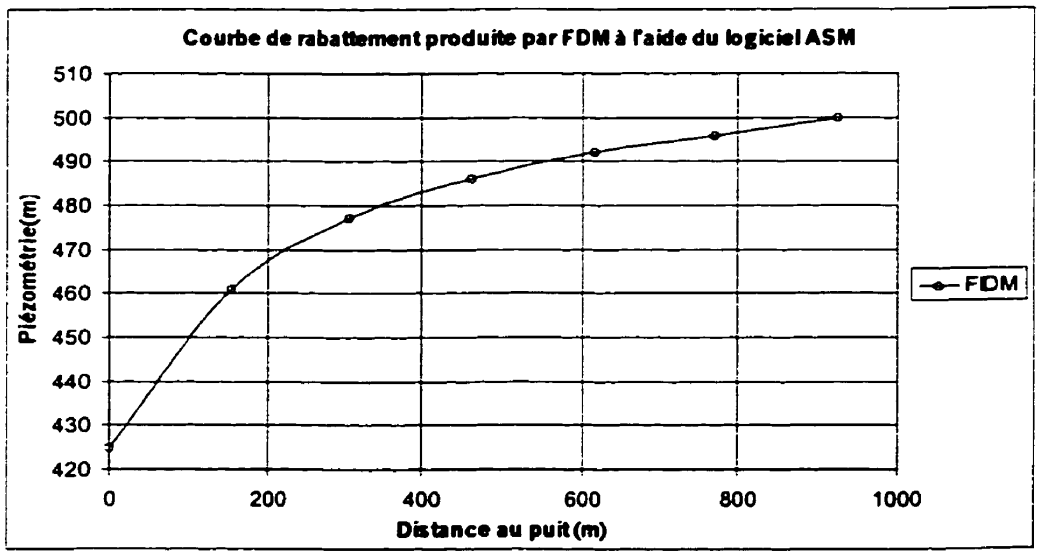

Figure 7-8 Courbe de rabattement obtenue à l'aide du logiciel ASM approximant le scénario de la nappe à écoulements convergents.

Pour faciliter la comparaison entre chacune des méthodes que nous mettons à l'essai, nous avons rapporté les résultats pour un rayon émergeant du puits et recoupant la périphérie sur une figure en coupe (c.f. figure 7.8), à la manière des courbes retournées par l'équation de Dupuit-Forchheimer (c.f. figure 7.5).

7.4 Le système logiciel proposé.

Le prototype de la IFDM pour les diagrammes Voronoï de points dans le plan que nous avons développé prend la forme d'un applet Java. L'algorithme de génération de la structure suit l'approche incrémentale (section 5.2.2) et repose sur la structure de donnée QuadEdge (section 7.1). L'ensemble du code source est fourni à l'annexe B. Nous donnons un aperçu du système d'un point de vue fonctionnel et organisationnel et passons en revue chacune de classes qui constituent le système, de quelle façon elles interagissent entre elles et pourquoi certains aspects du système se présentent sous une forme plutôt qu'une autre.

7.4.1 Classe ThreeD.

Cette classe représente en quelque sorte le point d'entrée du système puisqu'elle dérive de la classe java.applet.Applet. Elle est responsable de la réalisation en mémoire des différentes parties du système et de la capture des événements générés par la machine virtuelle Java et par l'usager. Elle orchestre l'ensemble des activités à la manière d'un contremaître.

7.4.2 Classe HydroExperiment et ses dérivés.

La classe HydroExperiment est une classe abstraite qui ne peut donc être réalisée en mémoire directement. Aucun objet créé en cours d'exécution n'est exclusivement de ce type. Les objets de ce type sont plutôt issus d'une classe dérivée de cette dernière. L'utilité d'une telle classe tient au fait qu'elle définit une interface et un comportement partagés par toutes les classes qui en dérivent. Dans la perspective de la réutilisation du code, la mise au point de classes abstraites représente donc parfois un exercice essentiel.

HydroExperiment encapsule la fonction qui calcule les solutions par relaxations successives. Ce solveur, bien qu'efficace, demeure très primitif étant donné qu'il ne s'agissait pas ici

d'innover en matière de calcul numérique. Quelques méthodes accessoires sont aussi définies dans la portion privée de l'interface.

De cette classe-mère, une seule classe a été dérivée dans le cadre de ce travail de recherche, classe encapsulant le comportement particulier de l'expérience à mener. Cette classe se nomme *DupuitForchheimer*, puisqu'il s'agit du scénario synthétique que nous tentons de reproduire. Elle définit la méthode *populate()*, déclarée au niveau de la classe-mère. Cette fonction est responsable de la construction du domaine d'écoulement : elle calcule la position des nœuds à insérer dans le diagramme Voronoï, de même qu'elle détermine leur type (frontière, charge libre, etc.) et leurs paramètres initiaux.

7.4.3 Classe Delaunay.

Cette classe est en fait le tessellateur Voronoï. Elle réalise l'algorithme incrémentale de Guibas et Stolfi et utilise aussi leur structure de données, le *QuadEdge*. La classe Delaunay est donc au cœur du système, bien qu'elle ignore totalement les fins pour lesquelles nous l'exploitons de par son caractère générique - ce qui en fait sa force. Son interface est relativement simple, laissant ses clients ajouter de nouveaux nœuds et défiler ses arêtes et de ses nœuds. Ces fonctionnalités sont exploitées par le solveur et aussi pour la construction du modèle 3D de l'applet.

7.4.4 Classe Cursor.

Un objet de la classe Cursor maintient en permanence la position du nœud le plus proche d'une position arbitraire dans le plan – position soumise par l'usager. Cet objet détermine aussi la surface de la cellule Voronoï qui serait générée par l'insertion de ce point et supporte la méthode d'interpolation dite par aires volées (*«natural neighbors interpolation»*). Cette composante du système n'est pas centrale à la résolution de notre problème et n'est donc pas utilisée dans le cadre de cette expérience.

7.4.5 Classe QuadEdge

La classe QuadEdge encapsule tout simplement la structure du même nom, de même que la majorité des méthodes qui lui sont associées. Quelques méthodes additionnelles lui ont

cependant été ajoutées pour supporter les simulations par IFDM. Bien que cette décision brise le caractère générique du QuadEdge, elle est justifiée au regard du nombre d'arêtes présentes dans un diagramme Voronoï.

En effet, la création d'une classe dérivée de QuadEdge définissant ces seules fonctions 'non-génériques' s'est montrée être un obstacle majeur à l'exécution du système sous Java. Les tables de pointeurs de fonctions, créées lors de la compilation des unités comprenant des classes dérivées, doivent en effet être traversées au moment de l 'appel des méthodes pendant l'exécution. Il s'agit d'un coût très élevé dans le contexte du langage Java, compte tenu de la position centrale qu'occupe le QuadEdge dans le système, problème probablement encore plus accentué par le fait qu'il s'agit d'un applet plutôt qu'une application autonome.

7.4.6 Classe Node

Les objets de la classe Node jouent plusieurs rôles dans le système. Ils supportent certaines des méthodes propres aux vecteurs 2D. Ils ont une position et sont capables de calculs vectoriels. Ils portent aussi une référence vers une charge hydraulique.

Cette dépendance est nécessaire, bien qu'intrusive, puisque ces charges portent les valeurs nodales du modèle hydrogéologique. Une solution à cette entorse pourrait être de forcer Node à manipuler des interfaces génériques vides, de type Attribut, de sorte qu'une modélisation portant sur autre chose que les charges hydrauliques n'aurait qu'à en supporter l'interface. Mais ce sont ici des considérations de design logiciel qui débordent la démonstration que nous cherchons à produire.

7.4.7 Classe Charge et ses dérivées

La classe Charge est une classe abstraite, à la manière de la classe HydroExperiment. Elle définit un nombre de méthodes statiques protégées qui ne sont invoquées que par les classes qui en dérivent, dans le but de tenir une forme de bilan hydrique. Elle comptabilise aussi la valeur du terme résiduel lors des itérations du solveur. Les classes dérivées réalisent quant à elles les comportements attendus des charges libres, des charges fixes et des frontières à gradient constant / nul.

Les charges libres sont celles où la piézométrie doit être évaluée par le solveur. À la fin de chaque itération, le solveur ajuste leur piézométrie, ce qui contribue en retour à l'évaluation du résiduel au niveau de la classe mère. Comme notre scénario ne dépend ni de la transmissivité hydraulique, ni du coefficient de storage, ces grandeurs n'ont été déclarées pour aucun des types de charge définis par ce système. Une solution générale devrait inclure ces grandeurs et en tenir compte lors des calculs du solveur.

Les charges fixes exhibent quant à elle des valeurs prédéterminées de la surface piézométrique. Ces charges sont utilisées partout où les eaux de surface sont en contact avec les eaux souterraines et aussi à tous les endroits où la surface piézométrique est connue et réputée en régime d'écoulement permanent. Ces cellules agissent à la manière d'une source d'eau inépuisable, et pour cette raison sont parfois simulées dans les systèmes commerciaux en assignant à des cellules à charge libre un coefficient de storage très très grand. Mais comme nous venons tout juste de le mentionner, notre simulation ne dépend pas de cette grandeur et il a donc été décidé d'ignorer cette avenue et de faire de ces cellules des entités à part entière du système.

Une frontière à gradient constant / nul est enveloppe entièrement une cellule de ce type. La position des interfaces dépend de la position des cellules voisines, coïncidant aux frontières Voronoï. Cette particularité vient du fait que l'approche que nous préconisons est une généralisation des différences finies à mailles centrée. Ainsi donc, le positionnement des interfaces imperméables ou à débit fixe pose un problème intéressant d'un point de vue fonctionnel parce qu'il implique l'opposition d'une cellule d'un type différent perpendiculairement à la frontière.

Le scénario qui a été développé à la section 7.2 ne nécessite pas le déploiement de charges à gradient nul, bien que la classe NullGradient soit définie, prête à être utilisée pour une autre étude de cas.

7.4.8 Classe CompGeom.

Cette classe regroupe un ensemble de fonctionnalités accessoires qui sont regroupées pour l'occasion sous le thème de la géométrie algorithmique. Ces fonctions servent par exemple à :

- déteminer de quel côté d'un segment un point est situé ;
- découvrir si trois points sont colinéaires ;
- confirmer si un point est dans un cercle ou hors de ce dernier ;
- calculer l'aire d'un triangle ;
- calculer le centre d'un cercle passant par trois points ;

7.4.9 Classes Model3D et Matrix3D.

Ces classes permettent de visualiser interactivement la surface piézométrique résultant de la simulation. Elles sont rudimentaires et ne permettent pas une inspection détaillée des surfaces générées. L'API Java3D aurait pu être mise à profit, mais après quelques essais infructueux, elle s'est avérée être très instable, causant souvent la suspension du système d'exploitation. Cette API n'est d'ailleurs toujours pas incluse au standard Java au moment où ces lignes sont écrites.

7.5 Simulation de la nappe à écoulements convergents par IFDM.

Pour tirer avantage du fait que les cellules de la IFDM peuvent avoir une taille variable, le maillage créé par la méthode *populate()* de la classe DupuitForschheimer suit un patron radial, où pour chacun des 12 rayons que compte le modèle, une progression géométrique est utilisée pour déterminer la position des nœuds du modèle. Cette progression suit la formule suivante, avec $r = 1.8$m. et $a_0 = 15$:

$$a_{i+1} = r * a_i$$

La frontière en périphérie du réseau sépare l'intérieur du modèle - où la piézométrie est inconnue – de l'extérieur. Cette frontière est constituée de charges fixes et est située à une distance de 928m. du centre de l'île où se trouve une cellule pratiquement circulaire de 15m. de rayon. Le maillage est présenté sur la figure 7.9.

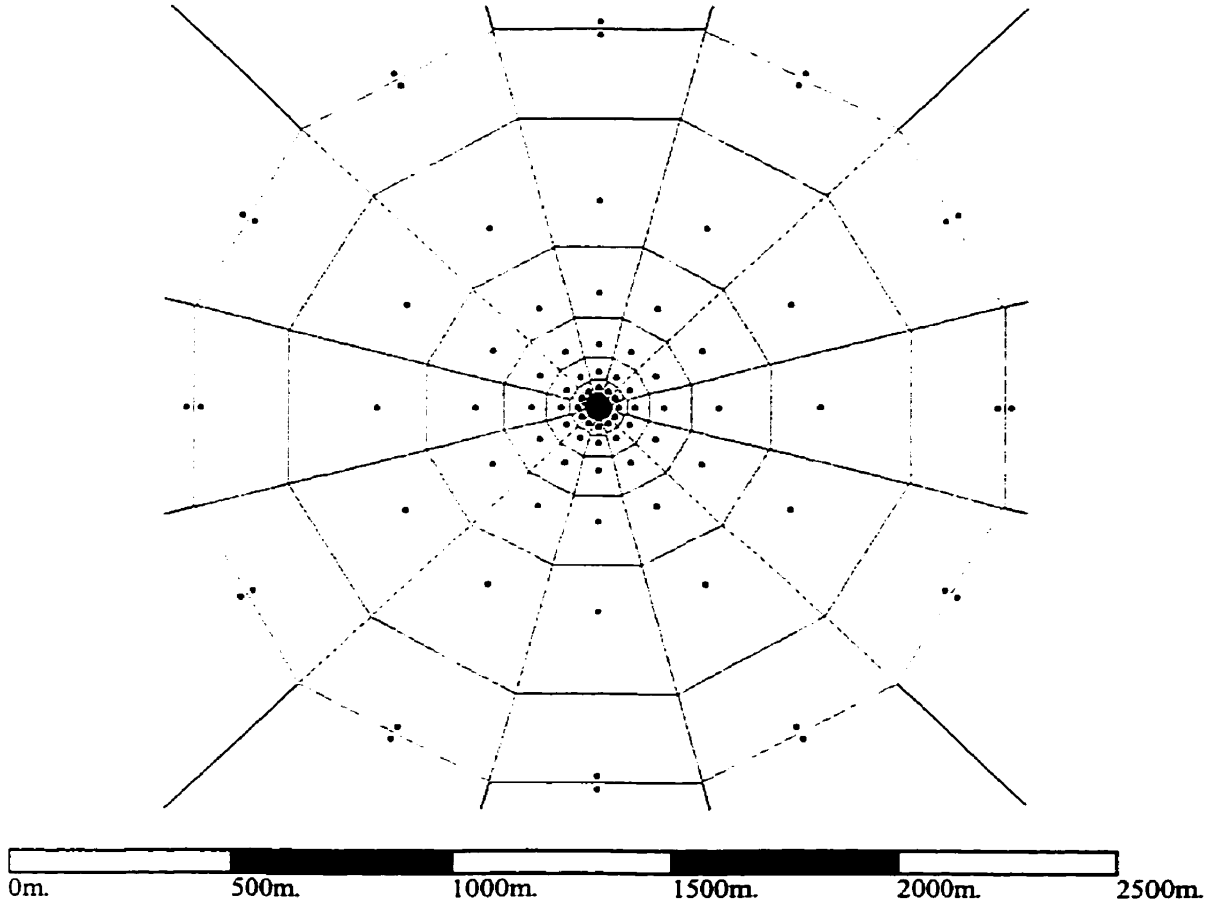

Figure 7-9 Vue en plan du modèle géométrique utilisé pour simuler la nappe à écoulements convergents. Les polygones du maillage sont en fait les cellules Voronoï associées aux nœuds du modèle.

La simulation résulte en une surface piézométrique que la figure 7.10 présente. Ces vues ont toutes été extraites directement à partir de l'applet Java. Cependant que le système de caméra virtuelle de l'applet ne permette pas une exploration très poussée de la surface, les résultats demeurent néanmoins intéressants.

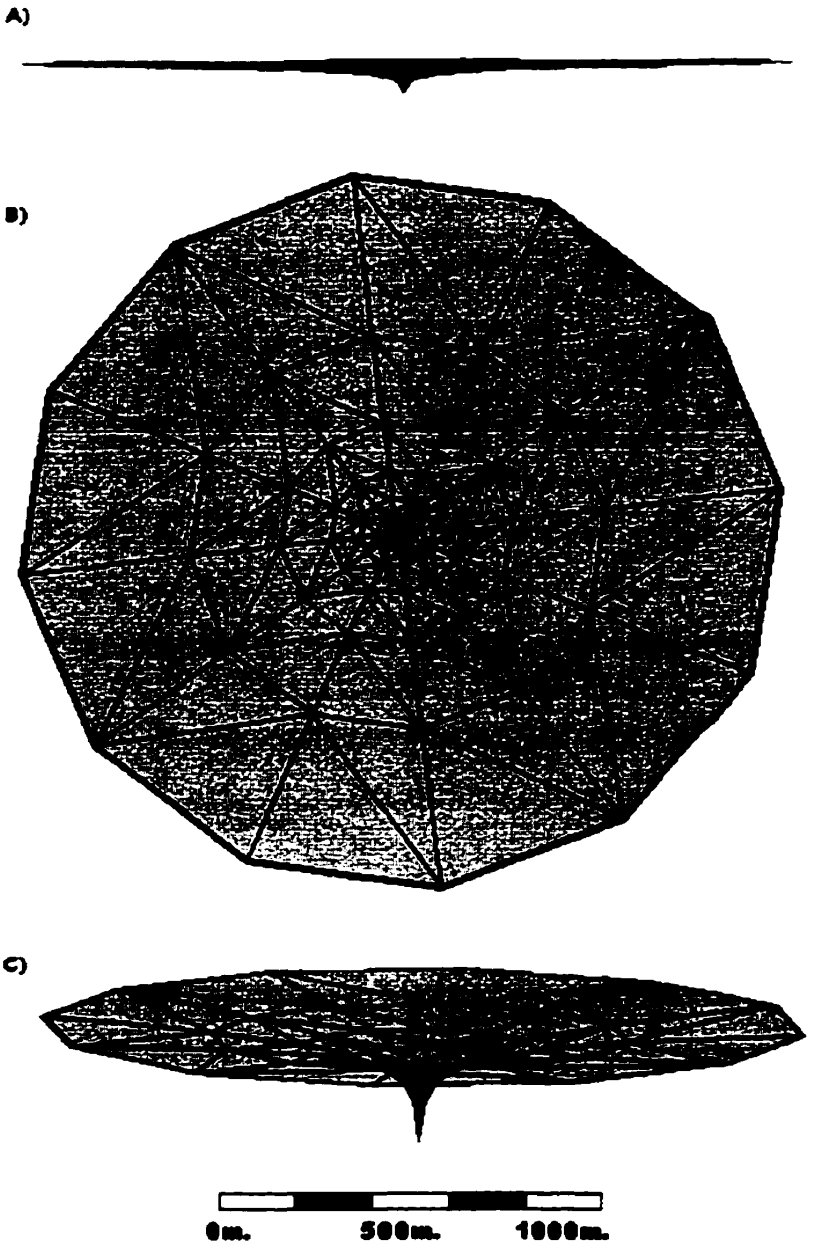

Figure 7-10 Vue en coupe a) et en plan b) du résultat de la simulation par différences finies intégrées pour la nappe à écoulements convergents. En c) est présentée une vue en perspective avec une exagération verticale égale à 4. Les triangles de ces trois constructions correspondent aux triangles Delaunay générés lors de la construction du diagramme Voronoï, la triangulation Delaunay étant la représentation duale de ce dernier.

La piézométrie de chacun des nœuds du modèle est dirigée vers la console Java une fois la condition de convergence atteinte, à la fin de la simulation. Encore une fois, afin de pouvoir comparer cette solution avec la solution analytique, une vue en coupe a été créée à partir de

ces données, plaçant le puits à l'origine, vue similaire à celle produite pour les courbes analytiques

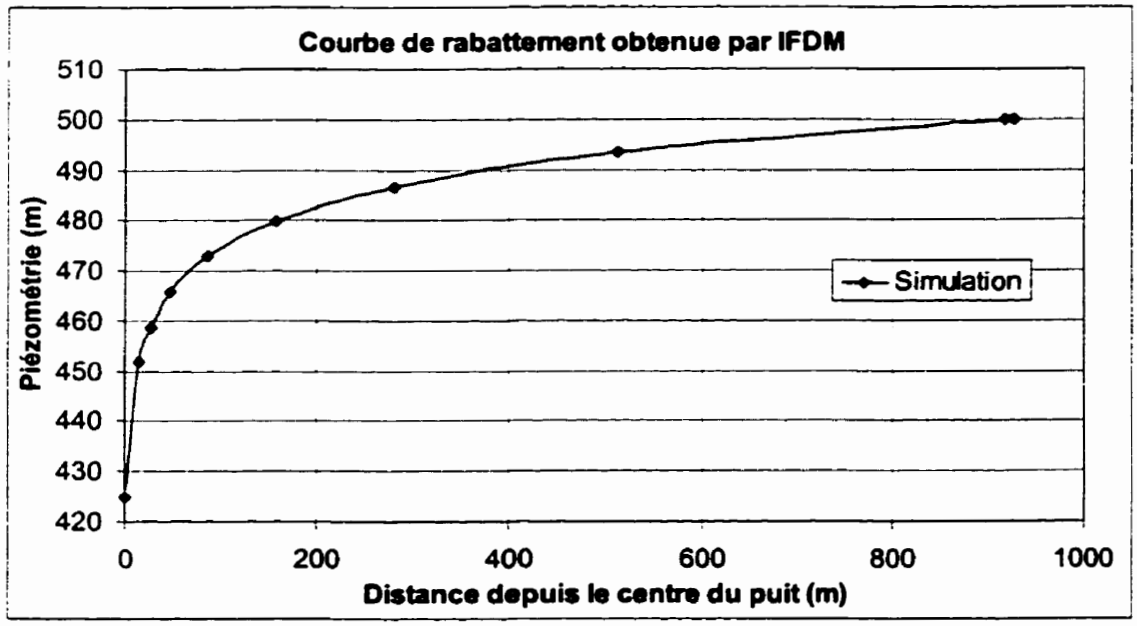

Figure 7-11 Courbe de rabattement de la nappe libre à écoulements convergents par IFDM.

7.6 Comparaison entre la IFDM, les solutions analytique et numérique.

Cette section complète la validation de la IFDM pour les nappes libres à écoulements convergents en comparant ses résultats avec ceux dérivés de l'équation de Dupuit-Forchheimer, de même qu'avec ceux produits par le logiciel commercial ASM, logiciel fondé sur la théorie des FDM.

Une comparaison graphique entre la IFDM et les courbes de la solution analytique est présentée sur la figure 7.12. Comme on peut le constater, la courbe de rabattement produite par la IFDM et celles produites par la solution de Dupuit-Forchheimer sont virtuellement identiques lorsque r, un paramètre qui ne peut être représenté explicitement dans un modèle discret, vaut 1.8m. Notons que plus la valeur de r est grande, plus la courbe tend à être déprimée. La table 7.1 présente les valeurs utilisées pour tracer ce graphique.

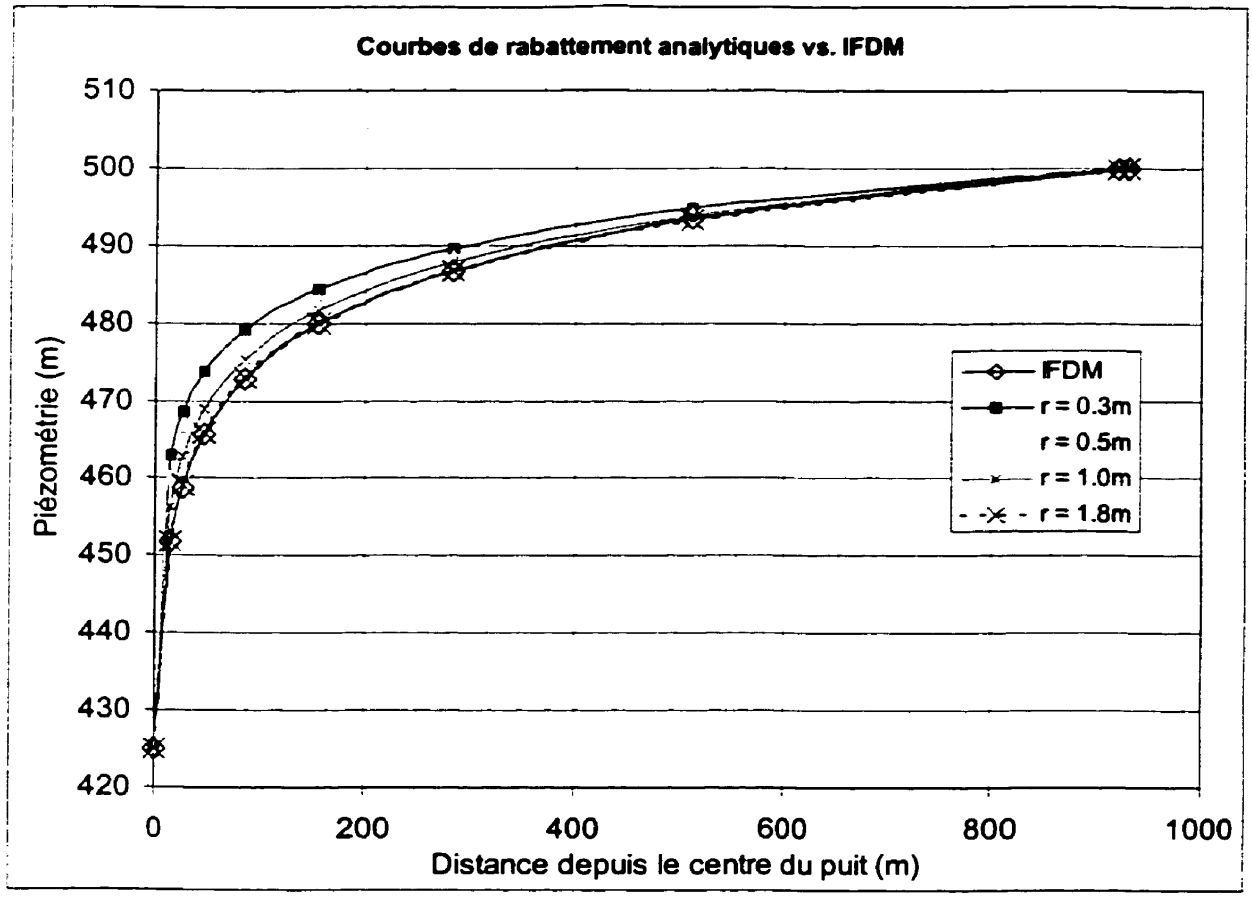

Figure 7-12 Comparaison entre le rabattement simulé et le rabattement attendu

Tableau 7-1 Valeurs de la piézométrie pour la simulation par IFDM et différentes évaluations de l'équation de Dupuit-Forchheimer

Distance au puits (m)	r = 0.3m	r = 0.5m	r = 1.0m	r = 1.8m	Simulation
0.00	425.00	425.00	425.00	425.00	425.00
15.00	463.03	460.41	456.20	451.86	451.81
27.00	468.47	466.26	462.70	459.03	458.79
48.00	473.74	471.91	468.97	465.94	465.72
87.00	479.13	477.68	475.36	472.97	472.88
157.00	484.42	483.34	481.62	479.86	479.84
283.00	489.64	488.93	487.79	486.63	486.70
510.00	494.81	494.45	493.88	493.31	493.39
918.00	499.91	499.90	499.89	499.88	499.88
928.00	500.00	500.00	500.00	500.00	500.00

Suivant une approche similaire, le graphique comparant la courbe calculée par IFDM à celle calculée par FDM est donné ci-après. Rappelons que ces modèles discrets ont tous deux 96 cellules à charge libre, soit des systèmes d'équations de même taille.

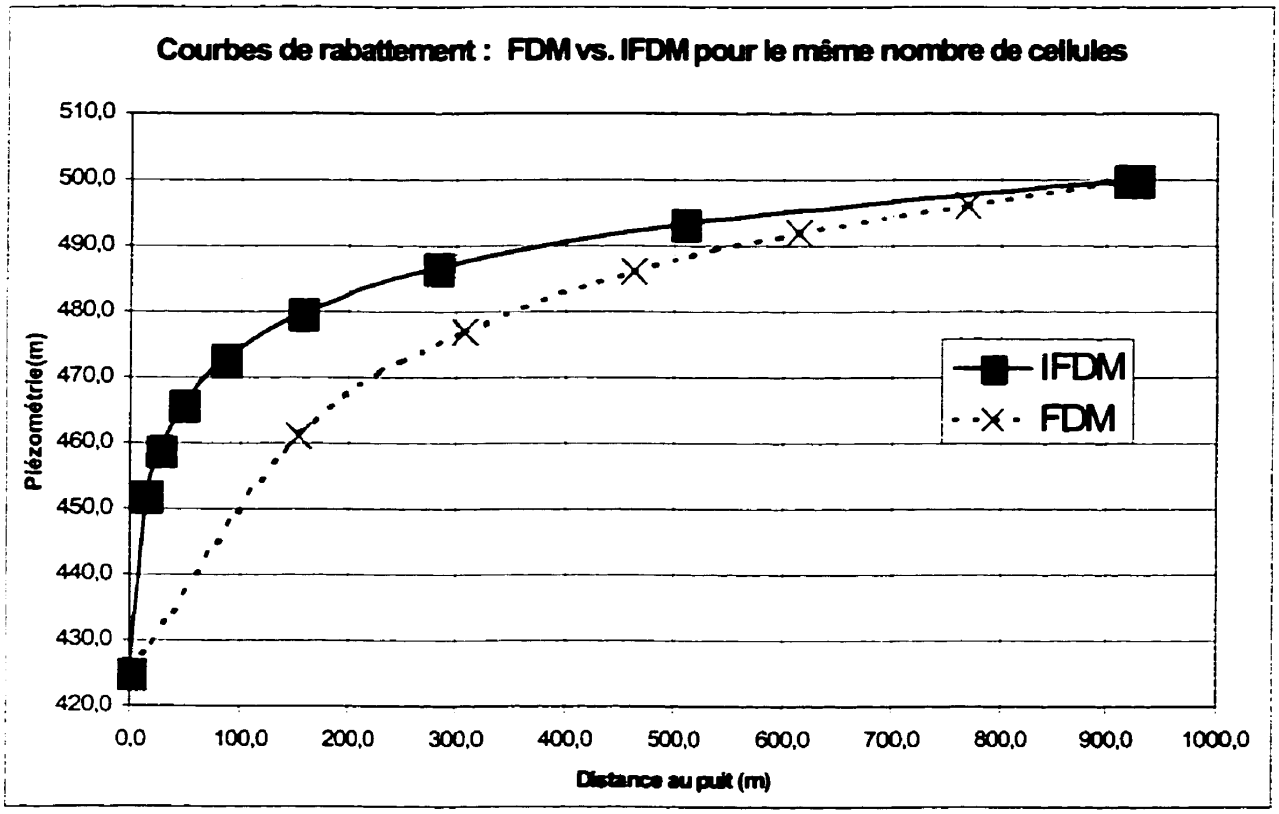

Figure 7-13 Courbes de rabattement de la FDM et de la IFDM.

Cette comparaison ne semble pas très avantageuse pour le logiciel ASM. La courbe produite présente un aspect très déprimé, ce qui laisse transparaître un artefact bien connu des spécialistes de traitement de l'image, le phénomène d'*«aliasing»*. Reprenant l'équation de Dupuit-Forchheimmer, si nous cherchons le rayon du puits r qui serait requis pour produire une courbe de rabattement de ce genre, la solution analytique nous retourne une valeur de 30m, comme le témoigne la figure 7.14. Autrement dit, le puits aurait un diamètre équivalent à la largeur d'un terrain de soccer !

Ce problème a d'ailleurs poussé la grande majorité des systèmes de modélisation hydrogéologique fondé sur la FDM à supporter les maillages à densité variable. Cependant, l'inconvénient des maillages de ce type vient du fait que pour accroître la densité au lieu d'un

puits, le domaine entier se voit traversé par des rangées et des colonnes de cellules de petite dimension, même très loin des sites d'intérêt, où de faibles gradients sont attendus. Ce phénomène se traduit par des systèmes d'équations de taille considérable.

La figure 7.14 fait le sommaire de cet exercice de validation en consolidant en un seul graphique les résultats obtenus de différentes manières pour le scénario de la nappe libre à écoulements convergents en régime d'écoulement permanent.

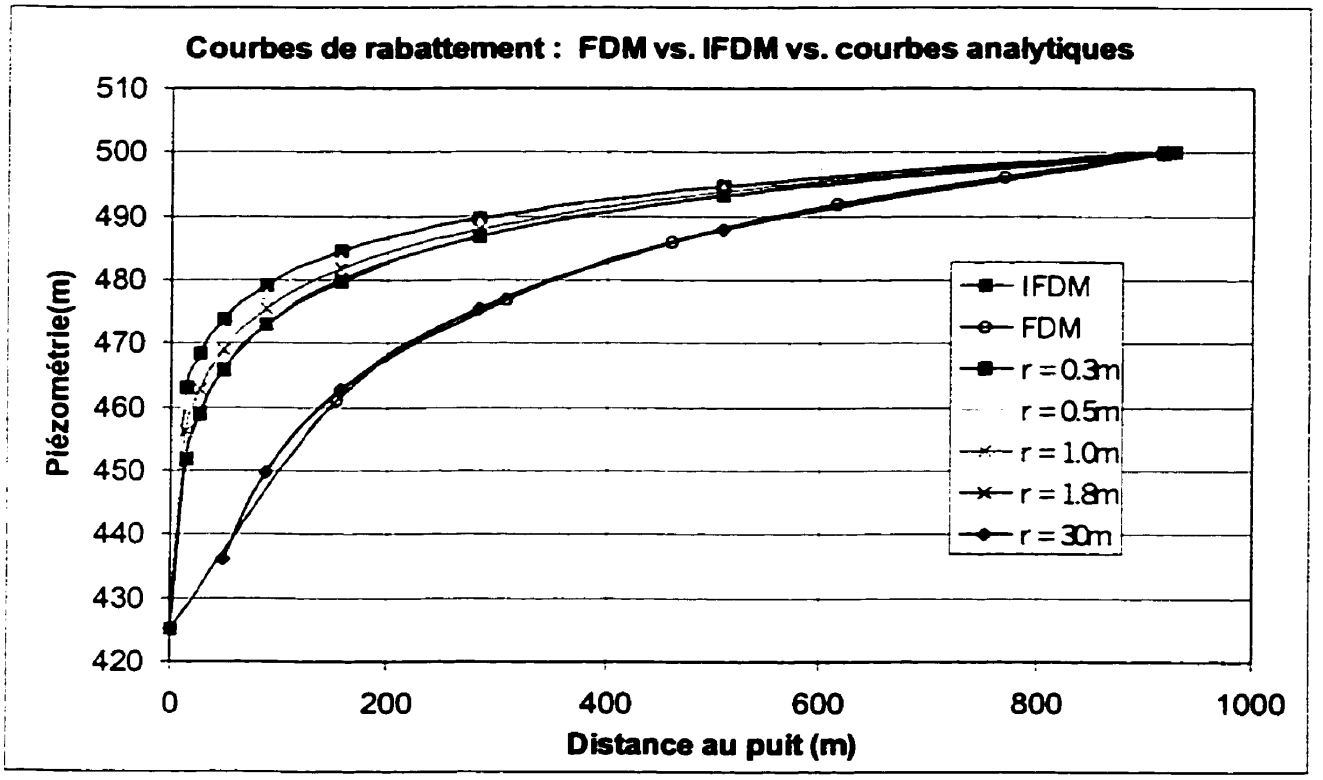

Figure 7-14 Courbe de rabattement de la IFDM comparée aux solutions analytiques calculées pour différentes valeurs de *r*, de même qu'à la solution de la FDM.

Est-ce que la IFDM sur un diagramme Voronoï de points donne de bons résultats ? Bien que cet exercice soit concluant, il ne fournit qu'une partie de la réponse à cette question. Plusieurs autres tests de validation, impliquant l'usage de frontières à débit constant / nul et des simulations en régime transitoire, devront être conduits avant de fournir une réponse plus complète, plus définitive.

8 Conclusions et Discussion.

Ce chapitre tire un ensemble de conclusions sur cette nouvelle approche et ouvre ensuite sur une discussion plus large portant sur l'impact qu'aura le couplage des outils de modélisation et de simulation sur le mode de gestion des données environnementales.

Quelques suggestions seront aussi émises quant à la nature et à la direction que pourraient prendre des travaux s'inscrivant dans le prolongement de ce travail de recherche.

8.1 Conclusions

Misant sur la simplicité des différences finies intégrées, nous avons tenté de mettre en évidence le potentiel du modèle spatial Voronoï comme structure de support à la simulation numérique des écoulements d'eau souterraine.

La comparaison des solutions du problème de la nappe libre à écoulements convergents en régime permanent données par le modèle analytique et la FDM conventionnelle, à celle obtenues par IFDM, nous indique que nous sommes sur la bonne voix. Le prototype que nous avons développé nous a donné une preuve concrète de son potentiel : pour un système d'équation de la même taille, la IFDM retourne une solution beaucoup plus précise que la FDM conventionnelle, à tout le moins pour des scénarios simples.

À ce stade de développement, les résultats se montrent fort encourageants. Nous avons réussi à démontrer qu'au prix d'une complexité mathématique relativement faible, l'adaptation du modèle Voronoï pour les applications de type «simulation» permettra de prolonger d'une façon très naturelle les fonctionnalités d'un SIG-Voronoï. Ces résultats semblent confirmer que le modèle spatial de Christopher Gold, développé selon l'approche géomatique, pourra supporter la conduite de simulations hydrogéologiques, ce qui constituera alors un apport original tant au domaine de la modélisation hydrogéologique qu'à celui des systèmes d'information géographique.

Tout indique qu'il serait possible de développer un SIG de type Voronoï supportant la génération de modèles géométriques et de simulations numériques directement à partir des

entités vectorielles peuplant les base de données géographiques. Cependant, le développement d'une solution de cette envergure suppose la disponibilité d'un SIG-Voronoï à architecture ouverte et le SIG-Voronoï de Christopher Gold est en phase de développement et ne montre pas à cette date la robustesse attendue d'une composante structurale aussi importante. Pour cette raison, le développement logiciel de la IFDM étendue demeure incomplet.

Finalement, ce travail vient partiellement valider les suppositions avancées dans le document *'Application Challenges to Computational Geometry'* publié en 1996 par le *'Computational Geometry Impact Task Force'*. Sans rompre radicalement avec les technologies traditionnelles de modélisation numérique, l'approche que nous suggérons repose sur un modèle géométrique alternatif, exploitant les propriétés géométriques du diagramme Voronoï généralisé dans le plan pour coupler plus intimement les aspects numériques et géométriques d'une simulation, et ce dans le cadre de problématiques environnementales très préoccupantes.

8.2 Avenues Futures.

Bien évidemment, une avenue très intéressante que des travaux pourraient emprunter dans le futur serait le développement d'un prototype pour des diagrammes Voronoï généralisés. Ce travail pourrait aboutir par la mise au point d'un système supportant la génération de modèles et de simulations directement à partir de bases de données géographiques vectorielles.

Outre cette application de la IFDM aux domaines souterrains, le récent gain de popularité du concept de «durabilité écologique» pousse plusieurs chercheurs à travailler au développement de solutions intégrées en matière d'environnement. En hydrogéologie, cette approche reconnaît que l'eau est une composante de l'environnement qui, pour être bien gérée, ne doit pas être décortiquée en «eaux de surface» d'un côté et «eaux souterraines» de l'autre. Cette tendance se traduira dans un futur proche par la mise œuvre d'outils de simulation intégrés mettant en relation un modèle numérique de surface et un modèle souterrain.

Or un examen des outils de modélisation courant révèle que la simulation des écoulements superficiels et souterrains modélisent incorrectement les interactions qui existent entre les deux réservoirs, partiellement parce que les modèles d'un réservoir simplifient à l'extrême les

mécanismes d'écoulement de l'autre, mais aussi parce qu'ils exploitent des structures de données spatiales distinctes, et souvent de natures tout à fait différentes (Genest et al, 1996).

Les problèmes posés par l'utilisation de structures de données spatiales différentes dans la représentation de la surface et du sous-sol pourraient aisément être résolus par l'utilisation de maillages partageant des caractéristiques géométriques communes. Au surplus, le partage d'un engin numérique pourrait permettre de modéliser en un seul exercice le cycle hydrologique continental sous divers régimes d'écoulement.

La IFDM et le modèle de données spatiales Voronoï pourraient bien constituer ensemble le cœur d'un outil de simulation intégré. Cet assemblage faciliterait la lecture des multiples scénarios émergeant de la phase de calibration des modèles et simplifierait la caractérisation du milieu par la sélection de jeux de paramètres réalistes. Nous croyons qu'un environnement SIG interactif intégré où les usagers auraient finalement la possibilité de simuler d'un trait les deux plus importants réservoirs d'eaux continentales pourrait être appelé à jouer un rôle déterminant dans les efforts de gestion des ressources hydriques.

8.3 Discussion.

De façon générale, bien que l'usager d'un SIG puisse intégrer bon nombre d'information, son jugement tombe rapidement dans l'arbitraire et le spéculatif dès que sa capacité ne lui permet plus de maîtriser l'ensemble des données nécessaires à la compréhension d'une réalité environnementale complexe. L'évaluation des solutions à un problème verse alors immanquablement dans l'irrationnel, invalidant à la source la procédure de prise de décision. C'est en partie pour contrer cette tendance et mieux apprécier certains processus physiques que les simulations numériques sont appelées à prendre une place prépondérante aux côtés des SIG, complétant la gamme des outils mis à la disposition des décideurs dans le cadre de vastes systèmes d'information à référence spatiale.

Malheureusement, beaucoup de SIG demeurent clos et n'ont pas les structures architecturales qui pourraient leur permettre de supporter des applications du genre que nous proposons ici. Ainsi il n'existe pas de SIG capable de répondre à des questions telles que : «Et si je draine le site d'enfouissement en excavant une tranchée à l'Est, est-ce que je risque d'assécher ces

terres agricoles ? ». Ou encore : «Où en seront rendues les réserves souterraines dans deux ans si la sécheresse continue ? ». L'aspect dynamique de certaines réalités environnementales demeure en effet fort mal accommodé par les SIG actuels.

Au tournant du millénaire, les simulations numériques sont appelées à occuper une place de premier plan auprès des gestionnaires du territoire. Il est impératif d'explorer les avenues qui mèneront à une meilleure représentation des phénomènes physiques dans le contexte des SIG afin de produire les vues synoptiques essentielles à la prise de décision.

Cet élan ne doit cependant d'aucune manière se limiter au développement de solutions boiteuses ou à la mise en œuvre de demi-solutions. Les spécialistes des SIRS et de la modélisation environnementale devront travailler de pair dans un exercice qui servira au mieux les communautés aux prises avec de sérieux problèmes de gestion des ressources naturelles.

9 Bibliographie.

Anderson, M. P., Woessner, W. W., Applied Groundwater Modeling : Simulation of Flow and Advective Transport, Academic Press, San Diego, 1992, 381pp.

Baker , C. P., Panciera, E. C., A Geographic information System for Groundwater System Planning, Journal of Soil and Water Conservation, v.45, pp246-248, 1990

Bear, J., Hydraulics of Groundwater, McGraw-Hill, New York, 1979, 567pp.

Boonstra, J., de Ridder, N.A., Numerical Modeling of Groundwater Basins : A User-Oriented Manual, International Institute for Land Reclamation and Improvement, Wageningen, 1981, 226pp.

Brémond, R., Contribution à l'Interprétation des Mesures de Débit et de Rabattement dans les Nappes Souterraines, Gauthier-Villars, Paris, 1965, 118pp

Chew, L. P., Chrisochoides, N., Sukup, F., Parallel Constrained Delaunay Meshing, AMD, v.220, Trends in Unstructured Mesh Generation, ASME, pp89-96, 1997

Computational Geometry Impact Task Force, Application Challenges to Computational Geometry, www.cs.princeton.edu/~chazelle/taskforce/CGreport.ps.Z, 1996, 57pp.

Deckers, F., Maintaining the Balance : A Groundwater Information System for the Netherland, GIS Europe, may 1995, pp32-34

Dussinberre, G. M., Heat-Tranfer Calculations by Finite Differences, International Textbook Company, Scranton, 1961, 302pp.

El-Kadi, A. I., Oloufa, A. A., Eltahan, A. A., Malik, H. U., Use of a Geographic Information System in Site-Specific Ground-Water Modeling, Ground Water, v.32, n.4, pp617-625, 1994

Emmons, H. W., The Numerical Solution of Partial Differential Equations, Quaterly of Applied Mathematics, v.2, n.2, p173-195, 1944

Genest, S., Kacem, M., Chevallier, J.-J., Lardin, P., Géovie-Tunisie : Vers une gestion intégrée des eaux de surface et des eaux souterraines, HydroMed 96 – Conférence sur la gestion des ressources hydriques du bassin Méditérranéen, Tunis, Tunisie, 1996

Gold, C.M., Spatial Data Structures – The Extension from One to Two Dimensions, dans Pau, L.F.(éditeur), « Mapping and Spatial Modeling for Navigation » (OTAN ASI Serie F no.65), Springer-Verlag, Brlin, Allemagne, pp11-39, 1990

Gold, C.M., The Interactive Map. dans: Advanced Geographic Data Modelling - Spatial Data Modeling and Query Languages for 2D and 3D Applications (ed. M. Molenaar and S. de Hoop), Netherlands Geodetic Commission Publications on Geodesy, n.40, pp121-128, 1994

Gold, C.M., Dynamic Spatial Data Structures : The Voronoi Approach, Proceedings of the Canadian Conference on GIS, Ottawa, March 1992, pp245-255

Gold, C. M., Edwards, G., The Voronoi Spatial Model - Two and Three Dimensional Applications in Image Analysis, ITC Journal, n.1, pp11-19, 1992

Gold, C.M., Roos, T., Surface Modelling With Guaranteed Consistency : An Object-Based Approach, IGIS'94, Geographic Information Systems (T. Roos, ed.), Lecture Notes in Computer Science No. 884, Springer-Verlag, Berlin, pp70-87, 1994.

Guibas, L., Stolfi, J., Primitives for the Manipulation of General Subdivision and thye Computation of Voronoi Diagrams, ACM Transactions on Graphics, v.4, n.2, pp74-123, 1985

Huyakorn, P. S., Lester, B. H., Faust, C. R., Finite Element Techniques for Modeling Groundwater flow in Fractured Aquifers, Water Resources Research, v.19, n.4, pp1019-1035, 1983

Imai, T., A Topology Oriented Algorithm For The Voronoi Diagram Of Polygons, CCCG'96 The Eighth Canadian Conference On Computational Geometry, Ottawa, August 12-15, 1996

Landry B., Mercier, M., Notions de Géologie, Modulo Éditeur, Québec, 2ième édition, 1992, 437pp.

MacNeal, R. H., An Asymmetrical Finite Difference Network, Quaterly of Applied Mathematics, v.11, n.3, p295-310, 1953

deMarsily, G., Quantitative Hydrogeology : Groudwater Hydrology for Engineers. Academic Press, Orlando, 1986, 440pp.

Narasimhan, T. N., Witherspoon, P. A., Numerical Model for Saturated-Unsaturated Flow in Deformable Porous Media : 1.Theory, Water Resources Research, v.13, n.3, p657-664, 1977

Narasimhan, T. N., Witherspoon, P. A., Numerical Model for Saturated-Unsaturated Flow in Deformable Porous Media : 1.Algorithm, Water Resources Research, v.14, n.2, p255-261, 1977

Narasimhan, T. N., Witherspoon, P. A., Numerical Model for Saturated-Unsaturated Flow in Deformable Porous Media : 3.Applications, Water Resources Research, v.14, n.6, p1017-1034, 1978

Orzol, L. L., McGrath, T. S., Summary of Modifications of the U.S. Geological Survey Modular, Finite-Difference, Ground-Water Flow Model to Read and Write Geographic Information System Files, Water Resources Bulletin, v.29, n.5, pp843-846, 1993

Rifai, H. S., Hendricks, L. A., Kilborn, K., Bedient, P. B., A Geographic Information System (GIS) user Interface for Delineating Wellhead Protection Areas, Grouind Water, v.31, n.3, pp480-488, 1993

Sasowsky, K. C., Gardner, T. W., Watershed Configuration and geographic Information System Parameterization for SPUR Model Hydrologic Simulations, Water Resources Bulletin, v.27, pp7-18, 1991

Southwell, R. V., Relaxation Methods in Theoretical Physics, Oxford Press, 1946

Thomas, R. G., Groudwater Models, Irrigation and Drainage Paper 21, FAO, Rome, 1973, 192pp.

Tyson, N. H., Weber, E. M., Use of Electronic Computers in the Simulation of the Dynamic Behavior of Groundwater Basins, Water Resources Engineering Conference, American Society of Civil Engineer, 13-17 mai 1963, Milwaukee

Tyson, N. H., Weber, E. M., Groudwater Management for the Nation's Future : Computer Simulation of Groundwater Basins, Proceedings of the American Society of Civeil Engineer, v.90, n.HY-94, 1964, p59-77

Yang, W., Gold, C.M., An Outline of a Dynamic Solution to Spatio-Temporal Windowing in an Urban GIS, Proceedings of Advances in Urban Spatial Information and Analysis, Wuhan, October 1993, pp36-44

10 Annexe A.

La présente section vise à fournir plus de détails quant à la manipulation des paramètres α, β et χ d'un segment Delaunay entre chaque itération lors d'une simulation par IFDM. Comme la surface piézométrique d'une cellule Voronoï associée à un segment Delaunay doit varier de manière à respecter des contraintes spécifiques, il doit être possible de retracer à tout moment la valeur de ces paramètres. La figure 9.1 présente le cas général considéré pour la suite de cette démonstration.

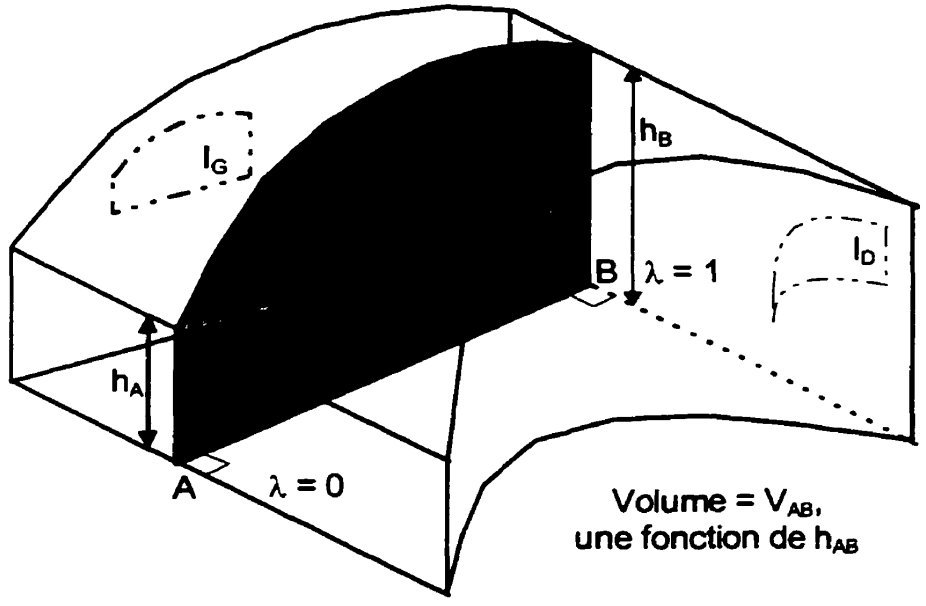

Figure 10-1 Distribution des paramètres physiques et géométriques dans le cas général pour l'évaluation des valeurs de α, β et χ lors d'une itération.

Comme le suggère cette figure, le choix d'une équation du second degré comme fonction d'interpolation servant à décrire la piézométrie sur AB est motivé par le fait que les contraintes h_A, h_B et V_{AB} sont des valeurs inhérentes à la résolution du problème, ce qui n'introduit aucun terme étranger à la résolution des paramètres α, β et χ. Le système d'équation à résoudre se présente donc sous la forme suivante :

$$h_A = \alpha\lambda^2 + \beta\lambda + \chi = \chi$$
$$h_B = \alpha\lambda^2 + \beta\lambda + \chi = \alpha + \beta + \chi$$
$$V_{AB} = \int_A^B h(\lambda) * l(\lambda) dL = |AB| \int_0^1 h(\lambda) * l(\lambda) d\lambda$$

Éq. 31

où $l(\lambda)$ exprime la distance dans le plan séparant les points $I_G(\lambda)$ et $I_D(\lambda)$. Dans le cas où les interfaces I_G et I_D seraient des fonctions du premier degré, V_{AB} de l'équation 25 devient :

$$I_G(\lambda) = \lambda r_B + (1-\lambda) r_A$$
$$I_D(\lambda) = \lambda s_B + (1-\lambda) s_A$$
$$l(\lambda) = I_D(\lambda) - I_G(\lambda) = \lambda(s_B - r_B) + (1-\lambda)(s_A - r_A)$$
$$l(\lambda) = \lambda(s_A - s_B - r_A + r_B) + (s_B - r_B)$$
$$l(\lambda) = \lambda u + v$$
$$V_{AB} = |AB| \int_0^1 h(\lambda) * l(\lambda) d\lambda$$
$$V_{AB} = |AB| \left(\frac{\alpha u}{4} + \frac{\alpha v + \beta u}{3} + \frac{\beta v + \chi u}{2} + \chi v \right)$$
$$V_{AB} = \alpha \left[\frac{|AB|}{12} * (3u + 4v) \right] + \beta \left[\frac{|AB|}{12} * (4u + 6v) \right] + \chi \left[\frac{|AB|}{12} * (6u + 12v) \right]$$

Éq. 32

Bien que ces expressions soient quelque peu longues, leur construction demeure relativement simple étant données les règles formelles selon lesquelles elles ont été établies.

Le développement de ces expressions suit le même modèle lorsque le segment est bordé par une multitude de voisins de part et d'autre. La différence majeure tient au fait que la mesure de la largeur de la cellule perpendiculairement au segment est dans ce cas une fonction composite prenant diverses formes selon le domaine de λ.

11 Annexe B.

Voici le code source, en langage Java, qui permet de conduire la simulation de l'écoulement des eaux souterraines par différences finies intégrées pour le scénario de la nappe à écoulements convergents. Il est à noter que ce prototype ne supporte que les diagrammes Voronoï de points dans le plan.

Fichier ThreeD.java

```
// Import directives
//----------------------
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;

// Class ThreeD
//-----------------
public class ThreeD
extends Applet
implements Runnable
{
	// Associations
	//-----------------
	private Model3D md;
	private boolean painted = true;
	private float xfac;
	private int prevx;
	private int prevy;
	private float xtheta;
	private float ytheta;
	private float scalefudge = 1;
	private Matrix3D amat = new Matrix3D();
	private Matrix3D tmat = new Matrix3D();
	private String mdname = null;
	private String message = null;


	// Manipulators
	//-----------------
	public
	void
	init()
	{
		amat.yrot( 20 );
		amat.xrot( 20 );
		resize(	size().width <= 20 ? 400 : size().width,
			size().height <= 20 ? 400 : size().height );
	}


	public
	void
	run()
	{
		// Instantiation of a delaunay triangulation
		//------------------------------------------
		Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
		Node n0 = new Node( 0, -10000 );
		Node n1 = new Node( -10000, 10000 );
		Node n2 = new Node( 10000, 10000 );
		Delaunay delaunay = new Delaunay( n0, n1, n2 );
```

```
        // Instantiation of a hydroExperiment
        //-----------------------------------
        HydroExperiment hydroExperiment = new DupuitForchheimer( delaunay,
        2000 );
        hydroExperiment.populate();
        hydroExperiment.relax();
        Model3D m = new Model3D();
        m.addVerts( delaunay.getVerts() );
        m.addEdges( delaunay.getEdges() );
        //-----------------------------------
        md = m;
        m.findBB();
        float xw = m.xmax - m.xmin;
        float yw = m.ymax - m.ymin;
        float zw = m.zmax - m.zmin;
        if( yw > xw )
        {
              xw = yw;
        }
        if( zw > xw )
        {
              xw = zw;
        }
        float f1 = size().width / xw;
        float f2 = size().height / xw;
        xfac = 0.7f * (f1 < f2 ? f1 : f2) * scalefudge;
        repaint();
}


public
void
start()
{
        if( md == null && message == null )
        {
              new Thread( this ).start();
        }
}


public
void
stop()
{
        //....
}


public
boolean
mouseDown( Event e, int x, int y )
{
        prevx = x;
        prevy = y;
        return true;
}


public
boolean
mouseDrag( Event e, int x, int y )
{
        tmat.unit();
        float xtheta = ( prevy - y ) * 360.0f / size().width;
        float ytheta = ( x - prevx ) * 360.0f / size().height;
        tmat.xrot( xtheta );
        tmat.yrot( ytheta );
        amat.mult( tmat );
        if( painted )
        {
              painted = false;
```

```
                repaint();
            }
            prevx = x;
            prevy = y;
            return true;
        }

        public
        void
        paint( Graphics g )
        {
            if( md != null )
            {
                md.mat.unit();
                md.mat.translate( -( md.xmin + md.xmax )/2,
                                  -( md.ymin + md.ymax )/2,
                                  -( md.zmin + md.zmax )/2 );
                md.mat.mult( amat );
                md.mat.scale( xfac, -xfac, 16*xfac/size().width );
                md.mat.translate( size().width/2, size().height/2, 8 );
                md.transformed = false;
                md.paint(g);
                setPainted();
            }
            else if( message != null )
            {
                g.drawString( "Error in model:", 3, 20 );
                g.drawString( message, 10, 40 );
            }
        }

        private
        synchronized
        void
        setPainted()
        {
            painted = true;
            notifyAll();
        }
}
```

Fichier HydroGeo.java

```
// Import directives
//------------------------
import java.*;

// Class HydroExperiment
//--------------------
abstract class HydroExperiment
{
        // Association
        //-----------------
        private Delaunay mDelaunay;
        private static final double sEps = 10.;

        // Constructor
        //-----------------
        public
        HydroExperiment( Delaunay delaunay )
        {
            mDelaunay = delaunay;
        }

        // Manipulators
```

```
//----------------
abstract
public
void
populate();

public
void
relax()
{
	// while the changes between successive iterations is greater than
	// a threshold value, traverse the list of all the edges to
	// calculate the adjustment that must be applied to the
	//  piezometric surface.

	System.out.println( "---------- SIMULATION -----------" );
	System.out.println( "" );

	// Start with the calculation of the conductance over each edge
	int numEdges = mDelaunay.mEdges.size();
	for( int i = 0; i < numEdges; i += 4 )
	{
		Edge e = ( Edge ) mDelaunay.mEdges.elementAt( i );
		if( e.isValidEdge() )
		{
			e.computeConductance();
			System.out.println( "Edge[" + i + "].mConductance =  "
					+ e.getConductance() );
		}
	}

	// Then compute the area of the polygon surrounding each node
	for( int i = 0; i < numEdges; i += 4 )
	{
		Edge e = ( Edge ) mDelaunay.mEdges.elementAt( i );
		for( int j = 0; j < 2; ++j )
		{
			if( !( e.org().getCharge() instanceof NullGradient ) )
			{
				if( e.org().getAreaX2() == 0 )
				{
					long areaX2 = computeAreaX2( e );
					System.out.println( "Node[" + i
							+ "].mArea = " + areaX2 );
				}
			}

			// Prepare the next iteration, we will then
			// be looking at the other end of e...
			e = e.sym();
		}
	}

	int iter = 0;
	double residual = 0.;
	int numNodes = mDelaunay.mNodes.size();
	do
	{
		// Iterating over the edges to compute
		// the corresponding flows.

		Charge.resetResidual();
		for( int i = 0; i < numEdges; i += 4 )
		{
			Edge e = ( Edge ) mDelaunay.mEdges.elementAt( i );
			if( e.org().getCharge() instanceof NullGradient ||
			    e.dest().getCharge() instanceof NullGradient )
			{
				continue;
			}
			if( e.org().getCharge() instanceof FreeCharge ||
			    e.dest().getCharge() instanceof FreeCharge )
```

```
                        {       // Water moves along e

                                double volumeX2 = 0;
                                if( e.getConductance() != 0
                                        && e.getPermeability() != 0 )
                                {
                                        volumeX2 = computeVolumeX2( e );
                                }
                                e.org().getCharge().addToDeltaX2( volumeX2 );
                                e.dest().getCharge().addToDeltaX2( -volumeX2 );
                        }
                }

                // Evaluating the residual
                for( int i = 0; i < numNodes; ++i )
                {
                        Node c = ( Node ) mDelaunay.mNodes.elementAt( i );
                        if( c.getCharge() instanceof FreeCharge ||
                            c.getCharge() instanceof FixedCharge )
                        {
                                c.getCharge().resetDeltaX2();
                        }
                }
                residual = Math.abs( Charge.getResidual() );
                ++iter;
        }
        while( residual > sEps );

        System.out.println( "" );
        System.out.println( "==========================================");
        System.out.println( "Converged in " + iter + " iterations." );
        System.out.println( "==========================================" );
        System.out.println( "" );

        System.out.println( ".............water table............." );
        System.out.println( "" );
        for( int i = 0; i < numNodes; ++i )
        {
                Node c = ( Node ) mDelaunay.mNodes.elementAt( i );
                if( c.getCharge() instanceof FreeCharge ||
                        c.getCharge() instanceof FixedCharge )
                {
                        long X[] = c.getXY();
                        System.out.println( "Node( " + X[ 0 ] + ", " + X[ 1 ]
                                        + " ) = " + c.getCharge().getVolumeX2()
                                        / c.getAreaX2() );
                }
        }
}

protected
void
insertNode( Node n )
{
        mDelaunay.insertNode( n );
}

private
double
computeVolumeX2( Edge e )
{
        double relaxFactor = 0.05;
        double orgH = e.org().getCharge().getVolumeX2()
                        / e.org().getAreaX2();
        double destH = e.dest().getCharge().getVolumeX2()
                        / e.dest().getAreaX2();
        double deltaH = destH - orgH;
        double thicknessX2 = orgH + destH;
        return( e.getConductance() * thicknessX2 * deltaH * relaxFactor );
}

private
```

```
        long
        computeAreaX2( Edge e )
        {
                try
                {
                        Edge first = e;
                        Node n0 = e.org();
                        long sumAreaX2 = 0;
                        do
                        {
                                Node n1 = e.right();
                                Node n2 = e.left();
                                sumAreaX2 += CompGeom.areaX2( n0, n1, n2 );
                                e = e.onext();
                        }
                        while( e != first );

                        e.org().setAreaX2( sumAreaX2 );
                        double piezo = e.org().getCharge().getPiezo();
                        if( piezo != 0 )
                        {
                                e.org().getCharge().setVolumeX2( piezo * sumAreaX2 );
                        }
                        return sumAreaX2;
                }
                catch( Exception exception )
                {
                        return 0;
                }
        }
}


// Class DupuitForchheimer
//----------------------------
class DupuitForchheimer
extends HydroExperiment
{
        // Associations
        //---------------
        private Node mOrg;
        private double mRange;


        // Constructor
        //--------------
        public
        DupuitForchheimer( Delaunay delaunay, int extentXY )
        {
                super( delaunay );
                mRange = extentXY/2;
                mOrg = new Node( ( long )mRange, ( long )mRange );
        }


        // Accessor
        //-----------
        public
        void
        populate()
        {
                int numRays = 12;
                for( int ray = 0; ray < numRays; ++ray )
                {
                        double angle = 2. * Math.PI / ( double )( numRays )
                                * ( double )( ray );
                        double a = 15.;
                        double r = 1.8;
                        double nextVal = a;
                        double val = 0.;
                        Charge charge = null;
                        do
```

```
                {
                        val = nextVal;
                        Node pos = new Node( (long)(val * Math.cos( angle )),
                                             (long)(val * Math.sin( angle )) );
                        Node node = pos.add( mOrg );
                        charge = new FreeCharge( 500. );
                        node.setCharge( charge );
                        insertNode( node );
                        nextVal *= r;
                }
                while( nextVal < mRange );
                val += 10.;

                charge = new FixedCharge( 500. );
                Node pos = new Node( (long)(val * Math.cos( angle )),
                                     (long)(val * Math.sin( angle )) );
                Node node = pos.add( mOrg );
                node.setCharge( charge );
                insertNode( node );
        }

        Charge charge = new FixedCharge( 425. );
        mOrg.setCharge( charge );
        insertNode( mOrg );
    }
}
```

Fichier Delaunay.java

```
// Import directives
//--------------------
import java.util.*;
import java.awt.*;


// Class Delaunay
//-------------------
class Delaunay
extends Object
{
    // Associations
    //---------------
    public Vector mEdges;
    public Vector mNodes;
    private Cursor mCursor;
    private long mStamp;
    private Node mN0;
    private Node mN1;
    private Node mN2;
    private boolean mDrawDelaunay;
    private boolean mDrawVoronoi;
    private boolean mDrawConvexHull;
    private boolean mDrawCrust;
    private boolean mDrawSkeleton;
    private boolean mDrawCursor;


    // Constructor
    //---------------
    public
    Delaunay( Node n0, Node n1, Node n2 )
    {
        mEdges = new Vector();
        mNodes = new Vector();
        mCursor = new Cursor();

        mDrawDelaunay = false;
        mDrawVoronoi = true;
        mDrawConvexHull = false;
        mDrawCrust = false;
```

```
        mDrawSkeleton = false;
        mDrawCursor = false;

        if( CompGeom.leftOf( n0, n1, n2 ) )
        {       //      Swap a pair of node to get a cw enum

                mN0 = n0;
                mN1 = n2;
                mN2 = n1;
        }
        else
        {
                mN0 = n0;
                mN1 = n1;
                mN2 = n2;
        }

        // These three points are at infinity...
        mN0.setInf();
        mN1.setInf();
        mN2.setInf();

        restart();
}

// Manipulators
//-----------------
public
void
setDelaunayDrawingState( boolean isActivated )
{
        mDrawDelaunay = isActivated;
}

public
void
setVoronoiDrawingState( boolean isActivated )
{
        mDrawVoronoi = isActivated;
}

public
void
setConvexHullDrawingState( boolean isActivated )
{
        mDrawConvexHull = isActivated;
}

public
void
setCrustDrawingState( boolean isActivated )
{
        mDrawCrust = isActivated;
}

public
void
setSkeletonDrawingState( boolean isActivated )
{
        mDrawSkeleton = isActivated;
}

public
void
setCursorDrawingState( boolean isActivated )
{
        mDrawCursor = isActivated;
}

public
void
restart()
```

```
{
        mStamp = Long.MIN_VALUE;
        mEdges.removeAllElements();
        mNodes.removeAllElements();
        makeBoundingTriangle();
}

protected
Edge
makeQuadEdge( Node org, Node dest )
{
        Edge e[] = new Edge[ 4 ];
        for( int i = 0; i < 4; ++i )
        {       // Create the four edges

                e[ i ] = new Edge();
        }

        e[ 0 ].setRot( e[ 1 ] );
        e[ 1 ].setRot( e[ 2 ] );
        e[ 2 ].setRot( e[ 3 ] );
        e[ 3 ].setRot( e[ 0 ] );
        e[ 0 ].setOnext( e[ 0 ] );
        e[ 1 ].setOnext( e[ 3 ] );
        e[ 2 ].setOnext( e[ 2 ] );
        e[ 3 ].setOnext( e[ 1 ] );
                // Set initial relationships

        for( int i = 0; i < 4; ++i )
        {       // Add the edges to mEdges

                mEdges.addElement( e[ i ] );
        }

        e[ 0 ].setOrg( org );
        e[ 0 ].setDest( dest );
        return e[ 0 ];
}

protected
Edge
connect( Edge a, Edge b )
{
        Edge e = makeQuadEdge( a.dest(), b.org() );
        e.splice( a.lnext() );
        e.sym().splice( b );
        return e;
}

public
void
insertNode( Node node )
{
        if( isInBoundingTriangle( node ) == false )
        {
                return;
        }

        try
        {
                 locate( node );
        }
        catch( Exception e )
        {
                return;
        }

        mNodes.addElement( node );
        Edge e = mCursor.getEdge();
        if( CompGeom.colinear( node, e.org(), e.dest() ) == true )
        {       // 'node' falls exactly on an edge
```

```
                Edge t = e.oprev();
                deleteEdge( e );
                e = t;
        }

        // Connect 'node' to existing nodes in the graph
        Node StartPnt = e.org();
        Edge newEdge = makeQuadEdge( StartPnt, node );
        mCursor.setEdge( newEdge.sym() );
        newEdge.splice( e );
        do
        {
                newEdge = connect( e, newEdge.sym() );
                e = newEdge.oprev();
        }
        while( e.dest() != StartPnt );

        // Make necessary changes in the graph
        while( true )
        {
                Edge t = e.oprev();
                boolean inCircle = CompGeom.inCircle( node, e.org(),
                                                      t.dest(), e.dest() );
                boolean rightOf = CompGeom.rightOf( t.dest(), e.org(),
                                                    e.dest() );
                if( rightOf && inCircle )
                {       // 'node' is in the circle, swap the edge than compute
                        // the circumcenter.

                        e.swap();
                        Node left = CompGeom.getCircumCenter(
                                        e.org(), e.dest(), e.onext().dest() );
                        e.setLeft( left );
                        e.lnext().setLeft( left );
                        e.lprev().setLeft( left );
                        e = e.oprev();
                }
                else if( e.org() == StartPnt )
                {       // The circumcenter on the left can be computed

                        Node left = CompGeom.getCircumCenter(
                                        e.org(), e.dest(), e.onext().dest() );
                        e.setLeft( left );
                        e.lnext().setLeft( left );
                        e.lprev().setLeft( left );
                        return;
                }
                else
                {       // The circumcenter on the left can be computed

                        Node left = CompGeom.getCircumCenter(
                                        e.org(), e.dest(), e.onext().dest() );
                        e.setLeft( left );
                        e.lnext().setLeft( left );
                        e.lprev().setLeft( left );
                        e = e.onext().lprev();
                }
        }
}

public
void
locateCursor( Node node )
{
        if( isInBoundingTriangle( node ) == false )
        {
                return;
        }

        try
        {
                locate( node );
```

```
        }
        catch( Exception e )
        {
                return;
        }

        // Find e on the border of the cavity
        Edge e = mCursor.getEdge();
        boolean leftOf = CompGeom.leftOf(
                        e.onext().dest(), e.org(), e.dest() );
        boolean inCircle = CompGeom.inCircle(
                        node, e.org(), e.dest(), e.onext().dest() );
        while( leftOf && inCircle )
        {
                e = e.onext();
                leftOf = CompGeom.leftOf(
                        e.onext().dest(), e.org(), e.dest() );
                inCircle = CompGeom.inCircle(
                        node, e.org(), e.dest(), e.onext().dest() );
        }

        // Will contain the solution
        Vector polys = new Vector();

        // Start walking CW on the boundary of the cavity
        e = e.sym();
        Node first = e.org();
        long nearDist = CompGeom.distSqr( node, e.org() );
        do
        {
                // Create a new polygon
                Poly poly = new Poly();
                double piezo = e.org().getCharge().getVolumeX2()
                                / e.org().getAreaX2();
                poly.setAttr( piezo );

                // Vertices of the poly
                Node v = CompGeom.getCircumCenter(
                        e.org(), e.dest(), node );
                poly.addNode( v );
                do
                {
                        poly.addNode( e.left() );
                        e = e.onext();
                }
                while( CompGeom.inCircle(
                        node, e.org(), e.dest(), e.onext().dest() ) );
                v = CompGeom.getCircumCenter( e.dest(), e.org(), node );
                poly.addNode( v );

                // Prepare 'e' for the next poly
                long tmpDist = CompGeom.distSqr( node, e.org() );
                if( tmpDist <= nearDist )
                {
                        nearDist = tmpDist;
                        mCursor.setEdge( e );
                        if( nearDist == 0 )
                        {
                                return;
                        }
                }
                polys.addElement( poly );
                e = e.sym();
        }
        while( e.org() != first );
        mCursor.setPolys( polys );
}

protected
void
deleteEdge( Edge e )
{
```

```
        e.splice( e.oprev() );
        e.sym().splice( e.sym().oprev() );
        Edge eRot;
        for( int i = 0; i < 3; ++i )
        {
                eRot = e.rot();
                mEdges.removeElement( e );
                e = eRot;
        }
        mEdges.removeElement( e );
}

protected
void
makeBoundingTriangle()
{
        // Creating the "cosmic triangle" bounding the domain
        Edge e0 = makeQuadEdge( mN1, mN2 );
        Edge e2 = makeQuadEdge( mN1, mN0 );
        e0.splice( e2 );
        Edge e1 = connect( e0, e2.sym() );

        // Compute the center of the triangle n1 n2 n3
        Node cc = CompGeom.getCircumCenter( mN0, mN1, mN2 );
        e0.setRight( cc );
        e0.setLeft( cc );
        e1.setRight( cc );
        e1.setLeft( cc );
        e2.setRight( cc );
        e2.setLeft( cc );
        mCursor.setEdge( e0.sym() );
        mCursor.setPos( cc );
        NullGradient charge = new NullGradient();
        mN0.setCharge( charge );
        mN1.setCharge( charge );
        mN2.setCharge( charge );
}

protected
boolean
isInBoundingTriangle( Node node )
{
        return( CompGeom.inTriangle( node, mN0, mN1, mN2 ) );
}

protected
void
locate( Node node )
throws Exception
{
        ++mStamp;
        mCursor.setPos( node );
        Edge e = mCursor.getEdge();
        if( CompGeom.rightOf( node, e.org(), e.dest() ) )
        {
                e = e.sym();
        }

        while( true )
        {
                if( node.equals( e.org() ) )
                {
                        mCursor.setEdge( e );
                        throw( new Exception( "Locate :  'node' and e.org()
                                are coincident." ) );
                }
                else if( node.equals( e.dest() ) )
                {
                        mCursor.setEdge( e.sym() );
                        throw( new Exception( "Locate :  'node' and e.dest()
                                are coincident." ) );
                }
```

```
                else if( node.equals( e.onext().dest() ) )
                {
                        mCursor.setEdge( e.onext().sym() );
                        throw( new Exception( "Locate :  'node' and
                                e.onext().dest() are coincident." ) );
                }
                else if( e.testAndStamp( mStamp ) )
                {
                        throw( new Exception( "Locate :  local-walk
                                algorithm stuck in a loop." ) );
                }
                else if( !( CompGeom.rightOf(
                                node, e.onext().org(), e.onext().dest() ) ) )
                {
                        e = e.onext();
                }
                else if( !( CompGeom.rightOf(
                                node, e.dprev().org(), e.dprev().dest() ) ) )
                {
                        e = e.dprev();
                }
                else
                {
                        mCursor.setEdge( e );
                        return;
                }
        }
}

// Accessors
//--------------
public
boolean
getConvexHullDrawingState()
{
        return mDrawConvexHull;
}

public
boolean
getVoronoiDrawingState()
{
        return mDrawVoronoi;
}

public
boolean
getDelaunayDrawingState()
{
        return mDrawDelaunay;
}

public
boolean
getCrustDrawingState()
{
        return mDrawCrust;
}

public
boolean
getSkeletonDrawingState()
{
        return mDrawSkeleton;
}

public
boolean
getCursorDrawingState()
{
        return mDrawCursor;
}
```

```
	public
	Cursor
	getCursor()
	{
		return mCursor;
	}

	public
	Vector
	getVerts()
	{
		Vector vec = new Vector();
		int size = mNodes.size();
		int index = 0;
		for( int i = 0; i < size; ++i )
		{
			Node n = ( Node ) mNodes.elementAt( i );
			if( !n.isInf() )
			{
				n.setTag( index );
				vec.addElement( n );
				++index;
			}
		}

		return vec;
	}

	public
	Vector
	getEdges()
	{
		Vector vec = new Vector();
		int size = mEdges.size();
		for( int i = 0; i < size; ++i )
		{
			Edge e = ( Edge ) mEdges.elementAt( i );
			if( e.isValidEdge() )
			{
				vec.addElement( e );
			}
		}

		return vec;
	}

	public
	void
	draw( Graphics g )
	{
		int entries = mEdges.size();
		for( int i = 0; i < entries; i += 4 )
		{
			Edge e = ( Edge ) mEdges.elementAt( i );
			if( e.isValidEdge() )
			{
				// Draw Voronoi Diagram
				e.drawHydro( g );
			}
		}

		mCursor.draw( g );
	}
}
```

Fichier Cursor.java

```
// Import directives
//---------------------
```

```
import java.awt.*;
import java.util.*;


// Class Cursor
//----------------
class Cursor
extends Object
{
	// Associations
	//-----------------
	private Edge mE;
	private Node mPos;
	private Vector mPolys;

	// Constructor
	//--------------
	public
	Cursor()
	{
		mE = null;
		mPos = null;
		mPolys = null;
	}

	// Manipulator
	//---------------
	public
	void
	setPos( Node pos )
	{
		mPos = pos;
	}

	public
	void
	setEdge( Edge e )
	{
		mE = e;
		mPolys = null;
	}

	public
	void
	setPolys( Vector polys )
	{
		mPolys = polys;
	}

	// Accessors
	//-------------
	public
	Node
	getPos()
	{
		return mPos;
	}

	public
	Edge
	getEdge()
	{
		return mE;
	}

	public
	double
	areaX2()
	{
		double result = 0;
		Enumeration enum = mPolys.elements();
		while( enum.hasMoreElements() )
```

```
            {
                  Poly p = ( Poly ) enum.nextElement();
                  result += p.areaX2();
            }
            return result;
      }

      public
      double
      piezoInterpolation()
      {
            // Interpolates a value at mPos for the Attr surface passing
            // through the points (x, y, attr) in the triangulation.

            double avgAttr = 0;
            double totAreaX2 = areaX2();
            Enumeration enum = mPolys.elements();
            while( enum.hasMoreElements() )
            {
                  Poly p = ( Poly ) enum.nextElement();
                  avgAttr += ( double )( p.getAttr()*p.areaX2() )/totAreaX2;
            }
            return avgAttr;
      }

      public
      void
      draw( Graphics g )
      {
            g.setColor( Color.red );
            if( mPolys != null )
            {
                  Enumeration enum = mPolys.elements();
                  while( enum.hasMoreElements() )
                  {
                        Poly p = ( Poly ) enum.nextElement();
                        p.draw( g );
                  }
            }
            Node.setColor( Color.black );
            mPos.draw( g );
            Node.setColor( Color.green );
            mE.org().draw( g );
            Node.setColor( Color.black );
      }
}
```

Fichier QuadEdge.java

```
// Import directives
//------------------
import java.awt.*;


// Class Edge
//-----------
class Edge
extends Object
{
      // Associations
      //-------------
      private Node mData;        // Origin
      private Edge mNext;        // Next counterclockwise edge
      private Edge mRot;         // Dual edge
      private long mStamp;
      private static Color sColor = Color.cyan;
      private double mPermeability;
      private double mConductance;

      // Constructor
```

```
//----------------
public
Edge()
{
	mData = null;
	mNext = null;
	mRot = null;
	mStamp = 0;
	mPermeability = 1.;
	mConductance = 0.;
}

// Manipulators
//----------------
public
void
setPermeability( double p )
{
	mPermeability = p;
}

public
double
getPermeability()
{
	return mPermeability;
}

public
void
computeConductance()
{
	mConductance = 0.;
	Node org = org();
	Node dest = dest();
	if( org.getCharge() instanceof NullGradient ||
		dest.getCharge() instanceof NullGradient )
	{
		return;
	}

	if( org.getCharge() instanceof FixedCharge &&
		dest.getCharge() instanceof FixedCharge )
	{
		return;
	}

	Node right = right();
	Node left = left();
	double lengthSqr = ( double )CompGeom.distSqr( org, dest );
	double widthSqr = ( double )CompGeom.distSqr( right, left );
	if( lengthSqr > Double.MIN_VALUE )
	{
		mConductance=mPermeability * Math.sqrt(widthSqr/lengthSqr);
	}
	else
	{
		mConductance = Double.MAX_VALUE;
	}
}

public
double
getConductance()
{
	return mConductance;
}

public
static
void
setColor( Color color )
```

```
{
        sColor = color;
}

public
void
splice( Edge b )
{
        Edge aa = onext().rot();
        Edge bb = b.onext().rot();
        Edge tmp = onext();
        setOnext( b.onext() );
        b.setOnext( tmp );
        tmp = aa.onext();
        aa.setOnext( bb.onext() );
        bb.setOnext( tmp );
}

public
void
swap()
{
        Edge a = oprev();
        Edge b = sym().oprev();
        splice( a );
        sym().splice( b );
        splice( a.lnext() );
        sym().splice( b.lnext() );
        setOrg( a.dest() );
        setDest( b.dest() );
}

public
void
setOrg( Node c )
{
        mData = c;
}

public
void
setRight( Node c )
{
        mRot.mData = c;
}

public
void
setDest( Node c )
{
        mRot.mRot.mData = c;
}

public
void
setLeft( Node c )
{
        mRot.mRot.mRot.mData = c;
}

public
void
setRot( Edge e )
{
        mRot = e;
}

public
void
setOnext( Edge e )
{
        mNext = e;
```

```
}

public
boolean
testAndStamp( long stamp )
{
	if( mStamp != stamp )
	{
		mStamp = stamp;
		return false;
	}
	return true;
}

// Accessors
//-------------
public
Node
org()
{
	return mData;
}

public
Node
dest()
{
	return mRot.mRot.mData;
}

public
Node
right()
{
	return mRot.mData;
}

public
Node
left()
{
	return mRot.mRot.mRot.mData;
}

public
Edge
rot()
{
	return mRot;
}

public
Edge
sym()
{
	return mRot.mRot;
}

public
Edge
irot()
{
	return mRot.mRot.mRot;
}

public
Edge
onext()
{
	return mNext;
}
```

```
public
Edge
oprev()
{
	return mRot.mNext.mRot;
}

public
Edge
lnext()
{
	return mRot.mRot.mRot.mNext.mRot;
}

public
Edge
lprev()
{
	return mNext.mRot.mRot;
}

public
Edge
rnext()
{
	return mRot.mNext.mRot.mRot.mRot;
}

public
Edge
rprev()
{
	return mRot.mRot.mNext;
}

public
Edge
dnext()
{
	return mRot.mRot.mNext.mRot.mRot;
}

public
Edge
dprev()
{
	return mRot.mRot.mRot.mNext.mRot.mRot.mRot;
}

public
boolean
isValidEdge()
{
	Node og = org();
	Node dt = dest();
	if( og == null || og.isInf() || dt == null || dt.isInf() )
	{
		return false;
	}
	else
	{
		return true;
	}
}

public
boolean
isNotConvexHull()
{
	return( isValidEdge() &&
		onext().isValidEdge() &&
		oprev().isValidEdge() );
```

```
        }

        public
        boolean
        isCrust()
        {
                Node r = right();
                Node l = left();
                Node o = org();
                Node d = dest();
                boolean crustTest = !( CompGeom.inCircle( l, o, r, d ) );
                return crustTest;
        }

        public
        void
        draw( Graphics g )
        {
                g.setColor( sColor );
                long u[] = org().getXY();
                long v[] = dest().getXY();
                g.drawLine( (int)u[ 0 ], (int)u[ 1 ], (int)v[ 0 ], (int)v[ 1 ] );
        }

        public
        void
        drawHydro( Graphics g )
        {
                // Choose the appropriate color
                Charge orgCharge = org().getCharge();
                Charge dstCharge = dest().getCharge();

                // Select the appropriate color.
                if( orgCharge instanceof FreeCharge ||
                        dstCharge instanceof FreeCharge )
                {
                        if( orgCharge instanceof FixedCharge ||
                                dstCharge instanceof FixedCharge )
                        {
                                // A fixed charge boundary
                                g.setColor( Color.green );
                        }
                        else if( orgCharge instanceof NullGradient ||
                                dstCharge instanceof NullGradient )
                        {
                                // A no flow boundary
                                g.setColor( Color.yellow );
                        }
                        else
                        {
                                // An internal boundary
                                g.setColor( Color.magenta );
                        }
                }
                long l[] = left().getXY();
                long r[] = right().getXY();
                g.drawLine( (int)l[ 0 ], (int)l[ 1 ], (int)r[ 0 ], (int)r[ 1 ] );
                g.setColor( Color.black );
                org().draw( g );
                dest().draw( g );
        }
}
```

Fichier Node.java

```
// Import directives
//--------------------
import java.awt.*;
```

```
// Class Node
//--------------
class Node
extends Object
{
	// Associations
	//---------------
	private static final long sEps = 0;
	private static Color sColor = Color.black;
	private long[] mX;		// Point coordinates
	private boolean mIsInf;		// Is it bounding the domain
	private Charge mCharge;
	private long mAreaX2;
	private int mTag;

	// Constructors
	//-----------------
	public
	Node( long x, long y )
	{
		mX = new long[ 2 ];
		mX[ 0 ] = x;
		mX[ 1 ] = y;
		mIsInf = false;
		mCharge = null;
	}

	public
	Node( Node n )
	{
		mX = new long[ 2 ];
		mX[ 0 ] = n.mX[ 0 ];
		mX[ 1 ] = n.mX[ 1 ];
		mIsInf = n.mIsInf;
		mCharge = n.mCharge;
	}

	// Manipulator
	//---------------
	public
	static
	void
	setColor( Color color )
	{
		sColor = color;
	}

	public
	void
	setInf()
	{
		mIsInf = true;
	}

	public
	void
	setAreaX2( long areaX2 )
	{
		mAreaX2 = areaX2;
	}

	public
	void
	setCharge( Charge charge )
	{
		mCharge = charge;
	}

	public
	void
	setTag( int tag )
	{
```

```
        mTag = tag;
}

// Accessor
//-----------
public
long[]
getXY()
{
        return mX;
}

public
boolean
isInf()
{
        return mIsInf;
}

public
long
getAreaX2()
{
        return mAreaX2;
}

public
Charge
getCharge()
{
        return mCharge;
}

public
Node
add( Node n )
{
        return( new Node( mX[ 0 ] + n.mX[ 0 ],  mX[ 1 ] + n.mX[ 1 ] ) );
}

public
Node
sub( Node n )
{
        return( new Node( mX[ 0 ] - n.mX[ 0 ],  mX[ 1 ] - n.mX[ 1 ] ) );
}

public
long
normSqr()
{
        return( mX[ 0 ]*mX[ 0 ] + mX[ 1 ]*mX[ 1 ] );
}

public
boolean
equals( Node n )
{
        Node v = this.sub( n );
        return( v.normSqr() <= sEps );
}

public
Node
multScal( long scal )
{
        return( new Node( mX[ 0 ]*scal, mX[ 1 ]*scal ) );
}

public
Node
divScal( long scal )
```

```
        {
                return( new Node( mX[ 0 ]/scal, mX[ 1 ]/scal ) );
        }

        public
        long
        dot( Node n )
        {
                return( mX[ 0 ]*n.mX[ 0 ] + mX[ 1 ]*n.mX[ 1 ] );
        }

        public
        Node
        cross()
        {
                return( new Node( -mX[ 1 ], mX[ 0 ] ) );
        }

        public
        int
        getTag()
        {
                return mTag;
        }

        public
        void
        draw( Graphics g )
        {
                g.setColor( sColor );
                g.fillOval( (int)( mX[ 0 ] - 2 ), (int)( mX[ 1 ] - 2 ), 4, 4 );
        }
}
```

Fichier Charge.java

```
// Charge interface
//---------------------
abstract class Charge
{
        // Associations
        //---------------
        private static double sResidual = 0.;
        private static double sBoundaryFlow = 0.;
        private static double sError = 0.;

        // Manipulator
        //--------------
        public
        static
        void
        resetResidual()
        {
                sResidual = 0.;
                sBoundaryFlow = 0.;
                sError = 0.;
        }

        protected
        static
        void
        addToResidual( double res )
        {
                sResidual += Math.abs( res );
        }

        protected
        static
        void
        addToBoundaryFlow( double flow )
        {
```

```
        sBoundaryFlow += flow;
    }

    protected
    static
    void
    addToError( double error )
    {
        sError += error;
    }

    abstract
    public
    void
    addToDeltaX2( double deltaX2 );

    abstract
    public
    void
    setVolumeX2( double volumeX2 );

    abstract
    public
    void
    setProductionX2( double productionX2 );

    abstract
    public
    void
    resetDeltaX2();

    // Accessor
    //------------
    public
    static
    double
    getResidual()
    {
        return sResidual;
    }

    public
    static
    double
    getBoundaryFlow()
    {
        return sBoundaryFlow;
    }

    public
    static
    double
    getError()
    {
        return sError;
    }

    abstract
    public
    double
    getVolumeX2();

    abstract
    public
    double
    getPiezo();

    abstract
    public
    double
    getProductionX2();
}
```

```
// FixedCharge
//---------------
class FixedCharge
extends Charge
{
	// Associations
	//-----------------
	private double mFixedPiezo;
	private double mVolumeX2;

	// Constructors
	//------------------
	public
	FixedCharge( double fixedPiezo )
	{
		mFixedPiezo = fixedPiezo;
		mVolumeX2 = 0.;
	}

	// Manipulators
	//------------------
	public
	void
	addToDeltaX2( double volumeX2 )
	{
		addToBoundaryFlow( volumeX2 );
	}

	public
	void
	setVolumeX2( double volumeX2 )
	{
		mVolumeX2 = volumeX2;
	}

	public
	void
	setProductionX2( double productionX2 )
	{

	}

	public
	void
	resetDeltaX2()
	{

	}

	// Accessors
	//--------------
	public
	double
	getVolumeX2()
	{
		return mVolumeX2;
	}

	public
	double
	getPiezo()
	{
		return mFixedPiezo;
	}

	public
	double
	getProductionX2()
	{
		return 0;
	}
```

```
}

// FreeCharge
//--------------------
class FreeCharge
extends Charge
{
	// Associations
	//-----------------
	private double mPiezo;
	private double mVolumeX2;
	private double mDeltaX2;
	private double mProductionX2;

	// Constructors
	//------------------
	public
	FreeCharge( double piezo )
	{
		mPiezo = piezo;
		mVolumeX2 = 0.;
		mDeltaX2 = 0.;
		mProductionX2 = 0.;
	}

	// Manipulators
	//------------------
	public
	void
	addToDeltaX2( double volumeX2 )
	{
		mDeltaX2 += volumeX2;
	}

	public
	void
	setVolumeX2( double volumeX2 )
	{
		mVolumeX2 = volumeX2;
	}

	public
	void
	setProductionX2( double productionX2 )
	{
		mProductionX2 = productionX2;
	}

	public
	void
	resetDeltaX2()
	{
		mDeltaX2 += mProductionX2;
		mVolumeX2 += mDeltaX2;
		addToResidual( mDeltaX2 );
		addToBoundaryFlow( mProductionX2 );
		mDeltaX2 = 0.;
	}

	// Accessors
	//--------------
	public
	double
	getVolumeX2()
	{
		return mVolumeX2;
	}

	public
	double
	getPiezo()
	{
```

```
            return mPiezo;
        }

        public
        double
        getProductionX2()
        {
            return mProductionX2;
        }
}

// NullGradient
//---------------------
class NullGradient
extends Charge
{
        // Association
        //--------------
        private Charge mMirror;

        // Constructors
        //------------------
        public
        NullGradient()
        {
            mMirror = null;
        }

        public
        NullGradient( Charge charge )
        {
            mMirror = charge;
        }

        // Manipulators
        //------------------
        public
        void
        addToDeltaX2( double volumeX2 )
        {
            addToError( volumeX2 );
        }

        public
        void
        setVolumeX2( double volumeX2 )
        {

        }

        public
        void
        setProductionX2( double productionX2 )
        {

        }

        public
        void
        resetDeltaX2()
        {

        }

        // Accessors
        //--------------
        public
        double
        getVolumeX2()
        {
            return 0.;
        }
```

```
	public
	double
	getPiezo()
	{
		if( mMirror != null )
		{
			return mMirror.getPiezo();
		}
		return 0.;
	}

	public
	double
	getProductionX2()
	{
		return 0.;
	}
}
```

Fichier CompGeom.java

```
// Import directives
//--------------------
import java.util.*;


// Class CompGeom
//-------------------
class CompGeom
{
	public
	static
	Node
	getCircumCenter( Node n0, Node n1, Node n2 )
	{	// Computes the center of the circle defined by n0 n1 n2

		Node p = n0.sub( n1 );
		Node q = n0.sub( n2 );
		Node a = q.cross().multScal( n0.add( n1 ).dot( p ) );
		Node b = p.cross().multScal( n0.add( n2 ).dot( q ) );
		return( a.sub( b ).divScal( 2*CompGeom.areaX2( n0, n1, n2 ) ) );
	}

	public
	static
	long
	areaX2( Node n0, Node n1, Node n2 )
	{	// Computes twice the surface of the orientable
		// triangle defined by n0, n1, n2

		Node c0 = n1.sub( n0 );
		long u[] = c0.getXY();
		Node c1 = n2.sub( n0 );
		long v[] = c1.getXY();
		long result = u[ 1 ]*v[ 0 ] - u[ 0 ]*v[ 1 ];
		return result;
	}

	public
	static
	boolean
	colinear( Node n0, Node n1, Node n2 )
	{
		long result = CompGeom.areaX2( n0, n1, n2 );
		return( result == 0 );
	}

	public
	static
	boolean
```

```
        inCircle( Node n0, Node n1, Node n2, Node n3 )
        {
                Node c0 = n1.sub( n0 );
                long s[] = c0.getXY();
                Node c1 = n2.sub( n0 );
                long t[] = c1.getXY();
                Node c2 = n3.sub( n0 );
                long u[] = c2.getXY();
                long mat3X3[] = {   s[0], s[1], ( s[0]*s[0] + s[1]*s[1] ),
                                    t[0], t[1], ( t[0]*t[0] + t[1]*t[1] ),
                                    u[0], u[1], ( u[0]*u[0] + u[1]*u[1] ) };
                long result = CompGeom.det3X3( mat3X3 );
                return( result <= 0 );
        }

        public
        static
        long
        det3X3( long m[] )
        {
                long result = m[0]*( m[4]*m[8] - m[5]*m[7] )
                            - m[1]*( m[3]*m[8] - m[5]*m[6] )
                            + m[2]*( m[3]*m[7] - m[4]*m[6] );
                return result;
        }

        public
        static
        boolean
        rightOf( Node n0, Node n1, Node n2 )
        {
                long result = CompGeom.areaX2( n0, n1, n2 );
                return( result < 0 );
        }

        public
        static
        boolean
        leftOf( Node n0, Node n1, Node n2 )
        {
                long result = CompGeom.areaX2( n0, n1, n2 );
                return( result > 0 );
        }

        public
        static
        boolean
        inTriangle( Node node, Node n0, Node n1, Node n2 )
        {       // n0, n1 and n2 can be ordered cw or ccw

                boolean t0 = ( CompGeom.areaX2( node, n0, n1 ) > 0 );
                boolean t1 = ( CompGeom.areaX2( node, n1, n2 ) > 0 );
                boolean t2 = ( CompGeom.areaX2( node, n2, n0 ) > 0 );
                return( ( t0 == t1 ) && ( t0 == t2 ) );
        }

        public
        static
        long
        distSqr( Node n0, Node n1 )
        {
                Node n = n0.sub( n1 );
                return( n.normSqr() );
        }
}
```

11.1 Fichier Poly.java

```
// Import directives
//--------------------
import java.awt.*;
```

```
// Class Poly
//--------------
class Poly
extends java.awt.Polygon
{
	// Associations
	//----------------
	private double mAttr;

	// Constructor
	//---------------
	public
	Poly()
	{
		mAttr = 0.;
	}

	// Manipulators
	//-----------------
	public
	void
	setAttr( double attr )
	{
		mAttr = attr;
	}

	public
	void
	addNode( Node n )
	{
		long v[] = n.getXY();
		addPoint( ( int )v[ 0 ], ( int )v[ 1 ] );
	}

	// Accessor
	//-----------
	public
	double
	areaX2()
	{
		if( npoints < 3 )
		{
			return 0;
		}
		double areaX2 = 0;
		int maxIter = npoints - 1;
		for( int i = 0; i < maxIter; ++i )
		{
			areaX2 += xpoints[ i ]*ypoints[ i + 1 ] -
					xpoints[ i + 1 ]*ypoints[ i ];
		}

		areaX2 += xpoints[ npoints - 1 ]*ypoints[ 0 ] -
				xpoints[ 0 ]*ypoints[ npoints - 1 ];
		return areaX2;
	}

	public
	double
	getAttr()
	{
		return mAttr;
	}

	public
	void
	draw( Graphics g )
	{
		g.setColor( Color.red );
		g.fillPolygon( this );
```

```
                g.setColor( Color.black );
                g.drawPolygon( this );
        }
}
```

Fichier Model3D.java

```
// Import directives
//--------------------
import java.awt.*;
import java.util.*;

// Class Model3D
//------------------
class Model3D
{
        // Associations
        //---------------
        private float vert[];
        private int tvert[];
        private int nvert;
        private int maxvert;
        private int con[];
        private int ncon;
        private int maxcon;
        boolean transformed;
        Matrix3D mat;
        float xmin;
        float xmax;
        float ymin;
        float ymax;
        float zmin;
        float zmax;
        private static Color gr[];


        // Constructor
        //---------------
        public
        Model3D()
        {
                mat = new Matrix3D();
                mat.xrot( 20 );
                mat.yrot( 30 );
        }


        // Manipulators
        //------------------
        public
        void
        addVerts( Vector vec )
        {
                Enumeration enum = vec.elements();
                while( enum.hasMoreElements() )
                {
                        Node n = ( Node ) enum.nextElement();
                        long v[] = n.getXY();
                        float z = ( float )( n.getCharge().getVolumeX2() ) /
                                        ( float )( n.getAreaX2() );
                        addVert( ( float )v[ 0 ], ( float )v[ 1 ], z );
                }
        }

        public
        void
        addEdges( Vector vec )
        {
                Enumeration enum = vec.elements();
                while( enum.hasMoreElements() )
```

```
        {
                Edge e = ( Edge ) enum.nextElement();
                int org = e.org().getTag();
                int dest = e.dest().getTag();
                addLine( org, dest );
        }
}

public
int
addVert( float x, float y, float z )
{
        int i = nvert;
        if( i >= maxvert )
        {
                if( vert == null )
                {
                        maxvert = 100;
                        vert = new float[ maxvert*3 ];
                }
                else
                {
                        maxvert *= 2;
                        float nv[] = new float[ maxvert*3 ];
                        System.arraycopy( vert, 0, nv, 0, vert.length );
                        vert = nv;
                }
        }
        i *= 3;
        vert[ i ] = x;
        vert[ i + 1 ] = y;
        vert[ i + 2 ] = z;
        return ++nvert;
}

public
void
addLine( int p1, int p2 )
{
        int i = ncon;
        if( p1 >= nvert || p2 >= nvert )
        {
                return;
        }
        if( i >= maxcon )
        {
                if( con == null )
                {
                        maxcon = 100;
                        con = new int[ maxcon ];
                }
                else
                {
                        maxcon *= 2;
                        int nv[] = new int[ maxcon ];
                        System.arraycopy( con, 0, nv, 0, con.length );
                        con = nv;
                }
        }
        if( p1 > p2 )
        {
                int t = p1;
                p1 = p2;
                p2 = t;
        }
        con[ i ] = ( p1 << 16 ) | p2;
        ncon = i + 1;
}


public
void
```

```
transform()
{
	if( transformed || nvert <= 0 )
	{
		return;
	}
	if( tvert == null || tvert.length < nvert*3 )
	{
		tvert = new int[ nvert*3 ];
	}
	mat.transform( vert, tvert, nvert );
	transformed = true;
}


// Accessors
//--------------
public
void
paint( Graphics g )
{
	if( vert == null || nvert <= 0 )
	{
		return;
	}
	transform();
	if( gr == null )
	{
		gr = new Color[ 16 ];
		for( int i = 0; i < 16; ++i )
		{
			int grey = ( int )( 170*( 1 -
					Math.pow( i/15.0, 2.3 ) ) );
			gr[ i ] = new Color( grey, grey, grey );
		}
	}
	int lg = 0;
	int lim = ncon;
	int c[] = con;
	int v[] = tvert;
	if( lim <= 0 || nvert <= 0 )
	{
		return;
	}
	for( int i = 0; i < lim; ++i )
	{
		int T = c[ i ];
		int p1 = ( ( T >> 16 ) & 0xFFFF ) * 3;
		int p2 = ( T & 0xFFFF ) * 3;
		int grey = v[ p1 + 2 ] + v[ p2 + 2 ];
		if( grey < 0 )
		{
			grey = 0;
		}
		if( grey > 15 )
		{
			grey = 15;
		}
		if( grey != lg )
		{
			lg = grey;
			g.setColor( gr[ grey ] );
		}
		g.drawLine( v[ p1 ], v[ p1 + 1 ], v[ p2 ], v[ p2 + 1 ] );
	}
}

public
void
findBB()
{
```

```
                if( nvert <= 0 )
                {
                        return;
                }
                float v[] = vert;
                float xmin = v[ 0 ], xmax = xmin;
                float ymin = v[ 1 ], ymax = ymin;
                float zmin = v[ 2 ], zmax = zmin;
                for( int i = nvert*3; ( i -= 3 ) > 0; )
                {
                        float x = v[ i ];
                        if( x < xmin )
                        {
                                xmin = x;
                        }
                        if( x > xmax )
                        {
                                xmax = x;
                        }
                        float y = v[ i + 1 ];
                        if( y < ymin )
                        {
                                ymin = y;
                        }
                        if( y > ymax )
                        {
                                ymax = y;
                        }
                        float z = v[ i + 2 ];
                        if( z < zmin )
                        {
                                zmin = z;
                        }
                        if( z > zmax )
                        {
                                zmax = z;
                        }
                }
                this.xmax = xmax;
                this.xmin = xmin;
                this.ymax = ymax;
                this.ymin = ymin;
                this.zmax = zmax;
                this.zmin = zmin;
        }
}
```

Fichier Matrix3D.java

```
// Class Matrix3D
//--------------------
class Matrix3D
{
        // Associations
        //----------------
        float xx, xy, xz, xo;
        float yx, yy, yz, yo;
        float zx, zy, zz, zo;
        static final double pi = 3.14159265;

        // Constructor
        //---------------
        public
        Matrix3D()
        {
                xx = 1.0f;
                yy = 1.0f;
                zz = 1.0f;
        }
```

```
// Manipulators
//----------------
public
void
scale( float f )
{
	xx *= f;
	xy *= f;
	xz *= f;
	xo *= f;
	yx *= f;
	yy *= f;
	yz *= f;
	yo *= f;
	zx *= f;
	zy *= f;
	zz *= f;
	zo *= f;
}

public
void
scale( float xf, float yf, float zf )
{
	xx *= xf;
	xy *= xf;
	xz *= xf;
	xo *= xf;
	yx *= yf;
	yy *= yf;
	yz *= yf;
	yo *= yf;
	zx *= zf;
	zy *= zf;
	zz *= zf;
	zo *= zf;
}

public
void
translate( float x, float y, float z )
{
	xo += x;
	yo += y;
	zo += z;
}


public
void
yrot( double theta )
{
	theta *= ( pi/180 );
	double ct = Math.cos( theta );
	double st = Math.sin( theta );

	float Nxx = ( float )( xx*ct + zx*st );
	float Nxy = ( float )( xy*ct + zy*st );
	float Nxz = ( float )( xz*ct + zz*st );
	float Nxo = ( float )( xo*ct + zo*st );

	float Nzx = ( float )( zx*ct - xx*st );
	float Nzy = ( float )( zy*ct - xy*st );
	float Nzz = ( float )( zz*ct - xz*st );
	float Nzo = ( float )( zo*ct - xo*st );

	xo = Nxo;
	xx = Nxx;
	xy = Nxy;
	xz = Nxz;
	zo = Nzo;
```

```
        zx = Nzx;
        zy = Nzy;
        zz = Nzz;
}


public
void
xrot( double theta )
{
        theta *= ( pi/180 );
        double ct = Math.cos( theta );
        double st = Math.sin( theta );

        float Nyx = ( float )( yx*ct + zx*st );
        float Nyy = ( float )( yy*ct + zy*st );
        float Nyz = ( float )( yz*ct + zz*st );
        float Nyo = ( float )( yo*ct + zo*st );

        float Nzx = ( float )( zx*ct - yx*st );
        float Nzy = ( float )( zy*ct - yy*st );
        float Nzz = ( float )( zz*ct - yz*st );
        float Nzo = ( float )( zo*ct - yo*st );

        yo = Nyo;
        yx = Nyx;
        yy = Nyy;
        yz = Nyz;
        zo = Nzo;
        zx = Nzx;
        zy = Nzy;
        zz = Nzz;
}


public
void
zrot( double theta )
{
        theta *= ( pi/180 );
        double ct = Math.cos( theta );
        double st = Math.sin( theta );

        float Nyx = ( float )( yx*ct + xx*st );
        float Nyy = ( float )( yy*ct + xy*st );
        float Nyz = ( float )( yz*ct + xz*st );
        float Nyo = ( float )( yo*ct + xo*st );

        float Nxx = ( float )( xx*ct - yx*st );
        float Nxy = ( float )( xy*ct - yy*st );
        float Nxz = ( float )( xz*ct - yz*st );
        float Nxo = ( float )( xo*ct - yo*st );

        yo = Nyo;
        yx = Nyx;
        yy = Nyy;
        yz = Nyz;
        xo = Nxo;
        xx = Nxx;
        xy = Nxy;
        xz = Nxz;
}


public
void
mult( Matrix3D rhs )
{
        float lxx = xx*rhs.xx + yx*rhs.xy + zx*rhs.xz;
        float lxy = xy*rhs.xx + yy*rhs.xy + zy*rhs.xz;
        float lxz = xz*rhs.xx + yz*rhs.xy + zz*rhs.xz;
        float lxo = xo*rhs.xx + yo*rhs.xy + zo*rhs.xz + rhs.xo;
```

```
        float lyx = xx*rhs.yx + yx*rhs.yy + zx*rhs.yz;
        float lyy = xy*rhs.yx + yy*rhs.yy + zy*rhs.yz;
        float lyz = xz*rhs.yx + yz*rhs.yy + zz*rhs.yz;
        float lyo = xo*rhs.yx + yo*rhs.yy + zo*rhs.yz + rhs.yo;

        float lzx = xx*rhs.zx + yx*rhs.zy + zx*rhs.zz;
        float lzy = xy*rhs.zx + yy*rhs.zy + zy*rhs.zz;
        float lzz = xz*rhs.zx + yz*rhs.zy + zz*rhs.zz;
        float lzo = xo*rhs.zx + yo*rhs.zy + zo*rhs.zz + rhs.zo;

        xx = lxx;
        xy = lxy;
        xz = lxz;
        xo = lxo;

        yx = lyx;
        yy = lyy;
        yz = lyz;
        yo = lyo;

        zx = lzx;
        zy = lzy;
        zz = lzz;
        zo = lzo;
}


public
void
unit()
{
        xo = 0;
        xx = 1;
        xy = 0;
        xz = 0;
        yo = 0;
        yx = 0;
        yy = 1;
        yz = 0;
        zo = 0;
        zx = 0;
        zy = 0;
        zz = 1;
}


public
void
transform( float v[], int tv[], int nvert )
{
        float lxx = xx, lxy = xy, lxz = xz, lxo = xo;
        float lyx = yx, lyy = yy, lyz = yz, lyo = yo;
        float lzx = zx, lzy = zy, lzz = zz, lzo = zo;
        for( int i = nvert*3; ( i -= 3 ) >= 0; )
        {
                float x = v[ i ];
                float y = v[ i + 1 ];
                float z = v[ i + 2 ];
                tv[ i ] = ( int )( x*lxx + y*lxy + z*lxz + lxo);
                tv[ i + 1 ] = ( int )( x • lyx + y*lyy + z*lyz + lyo);
                tv[ i + 2 ] = ( int )( x • lzx + y*lzy + z*lzz + lzo);
        }
}


public
String
toString()
{
        return( "[" + xo + "," + xx + "," + xy + "," + xz + ";" + yo +
                "," + yx + "," + yy + "," + yz + ";" + zo + "," + zx + "," +
```

```
                    zy + "," + zz + "]" );
        }
}
```

Fichier Dupuit.html

```
<title>3D Model: Dupuit-Forchheimer</title>
<hr>
<applet
        code=ThreeD.class
        width=600
        height=600>
</applet>
<hr>
<a href="ThreeD.java">The source.</a>
```