TIPE :: Tomographie discrète et application à la reconstruction de surfaces

Table des matières

La tomographie est un ensemble de techniques visant à reconstruire l'intérieur d'un objet sans y avoir accès comme dans un scanner. La tomographie discrète étudie le cas très particulier où le nombre d'axes de projections est limité et où l'objet est une image discrète. Le champ d'action de la tomographie discrète est cependant important car elle sert là où les scanners traditionnels sont impuissants. Elle se généralise aussi à des applications autres, notamment dans les problèmes d'emplois du temps. Ma démarche a été de redémontrer les résultats généraux sur la tomographie discrète présentés ci-dessous et d'en déduire des algorithmes, puis de les appliquer, ensuite, à la reconstruction tomographique d'une surface.

1 Problème général

On se donne n et m les dimensions de l'image discrète, assimilée à une matrice de $\{0,1\}^{n*m}$, et k axes de projections sous la forme $P_{a,b}(c) = \{a*i+b*j = c, (i,j) \in [1,n]*[1,m]\}$. La projection de la matrice M selon l'axe $P_{a,b}$ est $(Card\{(i,j) \in P_{a,b}(c), M_{ij} = 1\})_{c\in\mathbb{Z}}$. Pour simplifier on se restreint généralement aux $c \in \mathbb{Z}$ tels que $P_{a,b}(c) \cap [1,n]*[1,m] \neq 0$. Dans la suite, si l'on a deux projections, on considère les projections $P_{1,0}$ et $P_{0,1}$ pour la troisième on considère en plus $P_{1,1}$, car on peut généraliser les résultats sur ces axes.

2 Résolution dans le cas de deux projections

Étant donné $n+m$ nombres : $(u_1,...,u_n)$ et $(v_1,...,v_m)$; on cherche donc une matrice M de $\{0,1\}^{n*m}$ telle que la somme des éléments de la i^{e} ligne fasse u_i tandis que celle de la i^{e} colonne fasse v_i c'est à dire :

$$\forall i \in [1,n] \sum_{k=1}^{m} M_{ik} = u_i, \forall i \in [1,m] \sum_{k=1}^{n} M_{ki} = v_i$$

On dit alors que M satisfait les deux projections u et v.

2.1 Des conditions nécessaires

Une première condition

Tout d'abord si une matrice M est solution on a :

$$\sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij} = \sum_{i=1}^{n} u_i = \sum_{j=1}^{m} v_j \tag{1}$$

Une deuxième condition

Étant donné les $(u_1,...,u_n)$, on n'aura pas, pour tous les $(v_1,...,v_m)$ qui satisfont la première condition, une solution : si on prend par exemple $n = m = 2$ et $(u_1,u_2) = (2,0)$, alors il est clair que l'on n'aura des solutions que pour $(v_1,v_2) = (1,1)$.

On note $\tilde{u_j} = Card(\{i, j \leq u_i\})$. $\sum_{k=1}^{j} \tilde{u}_k = \sum_{k=1}^{n} min(u_k, j)$ est donc supérieur au nombre maximum de 1 dans j différentes colonnes. En triant les $(v_k)_{k\in[1,m]}$ par ordre décroissant, on peut formuler la condition :

$$\forall j \in [1,m] \sum_{k=1}^{j} v_k \leq \sum_{k=1}^{j} \tilde{u_k} \tag{2}$$

2.2 Des conditions suffisantes

Les deux conditions nécessaires précédentes sont en fait suffisantes, cela a été montré en 1957 par Ryser, de manière constructive, par un algorithme.

L'algorithme de Ryser

Tout d'abord on applique à v la permutation σ qui trie v. Ensuite on travaille par récurrence en conservant toutes les conditions :

Si m = 0 : c'est bon.

Sinon : on choisit les v_m indices $(h_1, ..., h_{v_m})$ des éléments les plus grands de u. Dans la colonne n on place des 1 aux indices $(h_1, ..., h_{v_m})$. On pose, si $j \in \{h_k\}$ alors $\dot{u}_j = u_j - 1$, sinon $\dot{u}_j = u_j$. On construit les $(m-1)$ premières colonnes en ré-appliquant l'algorithme.

Enfin on applique σ^{-1} aux colonnes de la matrice ainsi construite.

Le code C++, présent en annexe, reprend cette idée mais de façon différente, pour avoir une bonne complexité.

3 Le théorème de Ryser

Si l'on considère le cas simple $n = m$ et $(u_1, ..., u_n) = (v_1, ..., v_n) = (1, ..., 1)$ il y a $n!$ solutions (les matrices de permutations). On n'a donc pas, en général, de solution unique.

Définition : Équivalence de matrices On dit que deux matrices sont équivalentes si elles ont les mêmes projections.

3.1 Les bascules

Soit M une matrice tel qu'il existe x_1, x_2, y_1, y_2 tels que : $M_{x_1y_1}$=$M_{x_2y_2}$=1, $M_{x_1y_2}$=$M_{x_2y_1}$=0. Alors une bascule (x_1, x_2, y_1, y_2) consiste à modifier M pour que $M_{x_1y_1}$=$M_{x_2y_2}$=0, $M_{x_1y_2}$=$M_{x_2y_1}$=1.

3.2 Toutes les matrices équivalentes sont liées par des bascules

Une bascule produit une matrice équivalente à la première, mais Ryser a montré réciproquement que pour deux matrices équivalentes, on pouvait trouver une suite de bascules qui transforme l'une en l'autre.

En effet : on considère deux matrices équivalentes N et M. On considère ensuite le graphe bipartite dont les nœuds sont d'un coté les abscisses $(x_i)_{i\in[1,n]}$ et de l'autre les ordonnés $(y_i)_{i\in[1,m]}$. Comme N et M sont équivalentes, on montre qu'un tel graphe possède un cycle et on choisit alors un cycle de taille minimale $(x_1, y_1, x_2, ..., y_{k-1}, x_k, y_k)$. Alors si un arc relie x_1 et y_2 (voir Fig. 1) alors ce cycle est non minimal. S'il n'en existe pas, alors la bascule (x_1, x_2, y_1, y_2) rapproche bien N et M pour la distance $|N - M|_1$

Si l'on considère le graphe dont les nœuds sont les matrices de $\{0,1\}^{n*m}$ et les arêtes sont les bascules. L'ensemble des solutions pour deux projections données correspond donc à une composante connexe de ce graphe (éventuellement vide).

4 Complexité générale du problème de reconstruction pour plus de deux projections

4.1 Définitions :

problème de décision : Un problème de décision est une question dont la réponse est déterminée par un algorithme qui répond VRAI ou FAUX.

la classe de problèmes $\mathcal{P}$: Un problème de décision D est dit dans $\mathcal{P}$ s'il existe un algorithme qui le résout et qui est de complexité polynomiale.

la classe de problèmes $\mathcal{NP}$: Un problème de décision D est dit dans $\mathcal{NP}$ si on a un polynôme Q et un algorithme $A \in \mathcal{P}$ tel que pour toute instance w de taille n du problème D, $D(w)$ est VRAI si et seulement si on a un certificat c_w tel que $|c_w| < Q(n)$ et $A(w, c_w)$ répond VRAI.

réduction polynomiale : Un problème A est dit polynomialement réductible à B, que l'on note $A \leq_{\mathcal{P}} B$, s'il existe un algorithme C de complexité polynomiale tel que pour toute instance w du problème A, la réponse à A pour l'instance w est la même que celle de B à $C(w)$.

$\mathcal{NP}$-Complet : Un problème A est dit $\mathcal{NP}$-Complet si $A \in \mathcal{NP}$ et si pour tout problème $B \in \mathcal{NP}$ on a $B \leq_{\mathcal{P}} A$

En considérant l'algorithme qui vérifie qu'une matrice est bien solution d'un ensemble de projections, on voit que le problème de la tomographie à k projections est bien dans $\mathcal{NP}$.

Si $A \in \mathcal{NP}$-Complet avec $A \leq_{\mathcal{P}} B$ alors par transitivité de $\leq_{\mathcal{P}}$, si $B \in \mathcal{NP}$, B est $\mathcal{NP}$-Complet.

4.2 $\mathcal{NP}$-Complétude de l'existence d'une reconstruction pour 3 projections

En 1971, Stephen Cook a prouvé le caractère $\mathcal{NP}$-Complet du problème SAT : on simule l'exécution d'une machine de Turing non déterministe grâce à des variables booléennes. On montre alors que 3-SAT est $\mathcal{NP}$-Complet, puis que 1-3-SAT l'est aussi, et enfin que l'existence d'une reconstruction pour trois projections l'est aussi, par une astucieuse représentation des formules 1-3-SAT sous forme d'un circuit logique.

4.3 $\mathcal{NP}$-Complétude du problème général de k projections

Pour montrer la $\mathcal{NP}$-Complétude d'un système de k projections non parallèles, on montre par récurrence sur $k \in \mathbb{N}, 3 \leq k$, qu'à une formule 1-3-SAT donnée, on peut associer k projections $(p_1, ..., p_k)$ de taille polynomiale en la formule, et qu'il existe une solution à ces k projections, si et seulement la formule est satisfiable.

5 Résolution dans le cas de trois projections

Le problème de l'existence de la reconstruction à trois projections étant $\mathcal{NP}$-Complet, on ne connait pas actuellement d'algorithmes polynomial permettant de le résoudre sur des machines déterministes. On recherche donc une reconstruction (problème plus difficile que celui de l'existence) par des méthodes probabilistes et méta-heuristiques.

5.1 L'algorithme du recuit simulé

L'algorithme du recuit simulé est une méta heuristique qui s'inspire, comme son nom le suggère, d'un procédé de la métallurgie : on alterne de longs refroidissements lents, avec de courts réchauffements. Algorithmiquement, cela correspond à maintenir un paramètre température que l'on fait décroitre tout au long de l'algorithme. À chaque étape on effectue un petit déplacement dans notre espace de recherche. Si la différence entre la note attribuée à la position précédente et celle de la nouvelle position, multipliée par un facteur aléatoire, est inférieure au paramètre de température alors la modification est acceptée.

Par une recherche dans l'espace des matrices de $\{0,1\}^{n*m}$

Dans cette version, un déplacement consiste à inverser un certain nombre d'éléments de la matrice, et la note attribuée à une position, donc à une matrice, est la somme des erreurs sur chaque projection.

Par une recherche dans l'espace des solutions pour deux des projections

Une autre idée consiste à dire qu'une matrice qui satisfait trois projections est avant tout une matrice qui satisfait deux des projections. On se fixe deux des projections. On explore le graphe des solutions pour ces deux projections. Un déplacement est donc une suite de bascules (arêtes de notre graphe) tandis qu'une position est une matrice qui satisfait ces deux projections. *L'algorithme programmé est trop long pour tenir en annexe, mais il sera apporté le jour de l'entretien.*

6 Application à la reconstruction de la surface d'un objet

6.1 Problème et méthode

Je me suis fixé un objet, l'éponge de Menger après 3 ou 4 itérations. Je n'ai gardé de l'éponge que les projections sur chaque tranche et j'ai alors alors appliqué mes algorithmes pour reconstruire chaque tranche séparément puis j'ai affiché le tout. Sur des reconstructions exactes je n'ai pas pu pousser l'algorithme de reconstruction à partir de trois projections sur l'éponge à 4 itérations.

Pour voir à quoi ressemble l'éponge de Menger voir annexe. Si j'ai choisi cet objet, c'est parce qu'il est difficile à reconstruire de par sa forme symétrique et à trous.

6.2 Résultats

Voir les annexes pour des reconstructions colorés de l'éponge de Menger.

6.3 Remarque

On remarque alors que si l'éponge reconstruite à partir de deux projections sur chaque tranche est reconnaissable, celle reconstruite à partir de trois est sensiblement mieux reconstruite. Ce qui n'est pas surprenant car elle a été reconstruite avec plus d'informations.

A Bibliographie

INF441 Modex informatique Programmation efficace Christoph Dürr

Traitement du signal 1995 - Volume 12 n°5 S . Reboul, A. Taleb-ahmed, M.m Rousset, E Wattrelot, J.P. Dubus

Thèse de Ph.D. de K.J. Batenburg (principalement chapitre 2)

Discrete Tomography : Foundations, Algorithms, And Applications G.T. Herman, A. Kuba, J.J. Benedetto

Advances in discrete tomography and its applications G.T. Herman, A. Kuba

B Algorithme de Ryser

L'algorithme tel qu'utilisé pour reconstruire une matrice à partir de deux projections :

```
#include <cstdio>
#include <deque>
#include <algorithm>
#include <vector>
using namespace std ;
vector< vector<bool> >grille;
vector< vector<int> >u ;
vector<pair<int,int> > v ;
int largeur, hauteur, lu;

int main()
{
  scanf("%d %d",&largeur,&hauteur ) ;
  for( int x = 0 ; x < largeur ; x++ )
    {
      scanf("%d",&lu);
      v.push_back( make_pair( lu, x)) ;
      u.push_back( vector<int>() ) ;
    }
  for( int y = 0 ; y < hauteur ; y++ )
    {
      scanf("%d",&lu) ;
      u[lu].push_back( y ) ;
    }
  for( int y = 0 ; y < hauteur ; y++ )
    grille.push_back( vector<bool> ( largeur , false ) ) ;
  sort( v.begin(), v.end()) ;

  for( int x = largeur-1 ; x >= 0 ; x-- )
    {
      vector<pair<int,int> > prend ;
      while( !((u.back()).size()) )
        u.pop_back();
      int cur_col = u.size()-1 ;

      while( v[x].first )
          if( u[cur_col].size() )
          {
            grille[ u[cur_col].back() ][ v[x].second ] = true;
            prend.push_back( make_pair( cur_col-1 , u[cur_col].back() ) ) ;
            u[cur_col].pop_back();
          }
          else
            cur_col--;
      for( size_t id = 0 ; id < prend.size() ; id++ )
        u[prend[id].first].push_back( prend[id].second ) ;
    }
  for( int y = 0 ; y < hauteur ; y++ ,printf("\n"))
    for( int x = 0 ; x < largeur ; x++ )
      printf("%d ",(int)grille[y][x] );
  return 0 ;
}
```

Si *sort*() utilise un tri panier (ou si $ln(hauteur) \leq largeur$), on est en $O(n * m)$, ce qui est aussi la taille de la sortie.

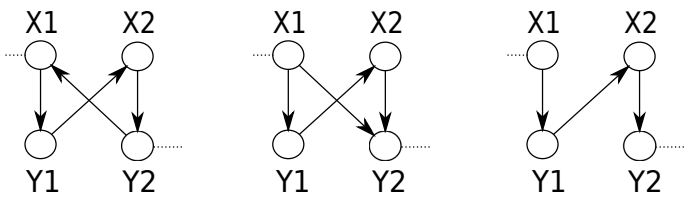
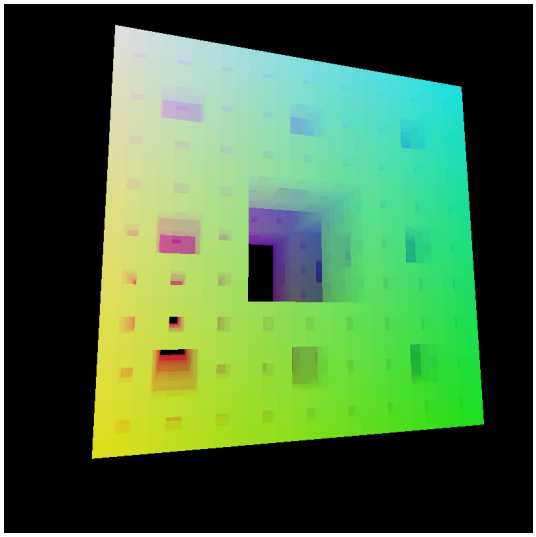
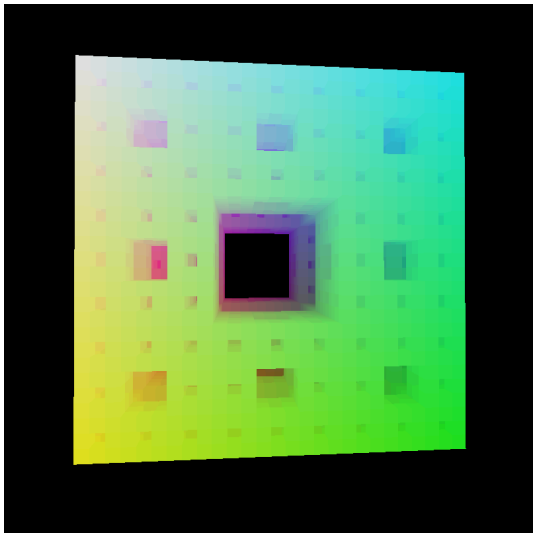


FIGURE 1 – La distinction des trois cas.

C Le modèle, l'éponde de Menger, pour 3 et 4 itérations.

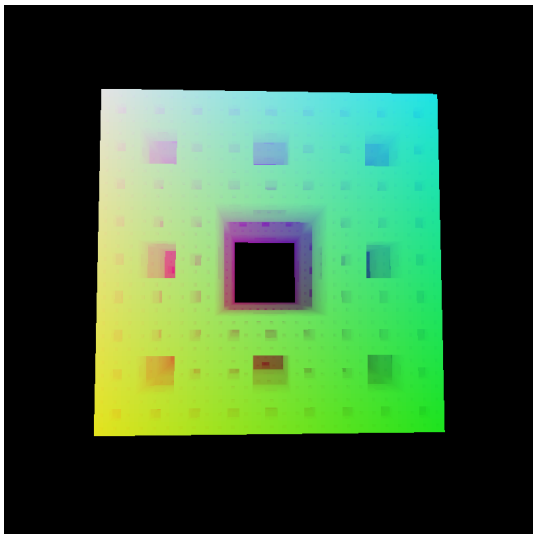
petit modèle

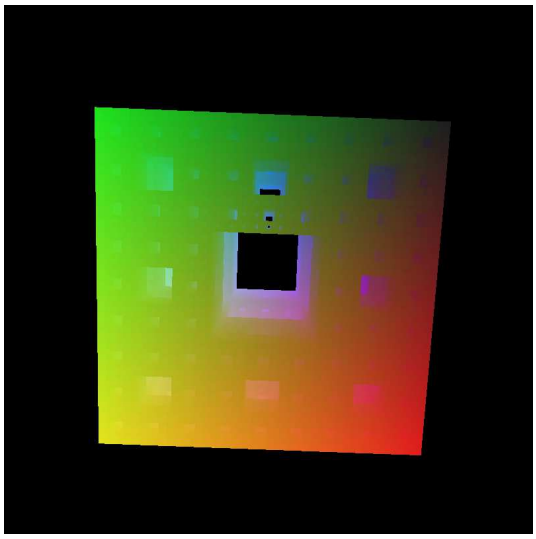




nombre de voxels : 19683

grand modèle

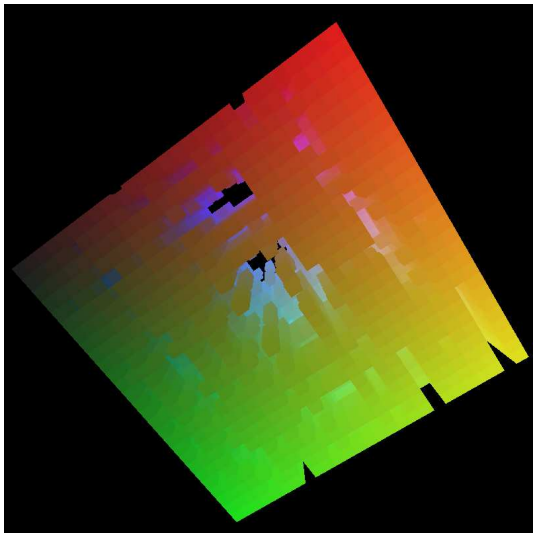


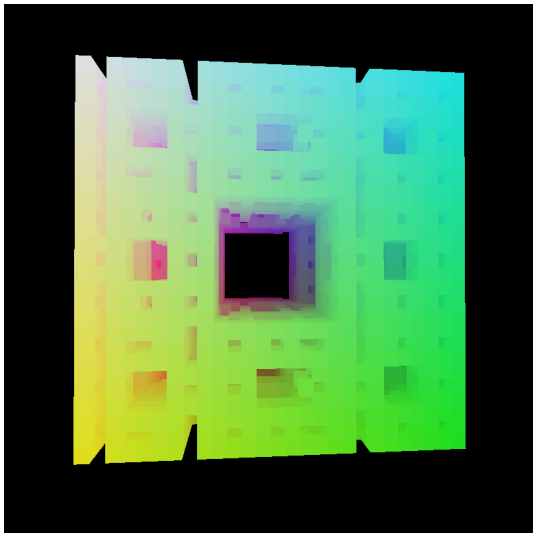


nombre de voxels : 531441

D Les reconstructions pour 3 et 4 itérations avec deux projections.

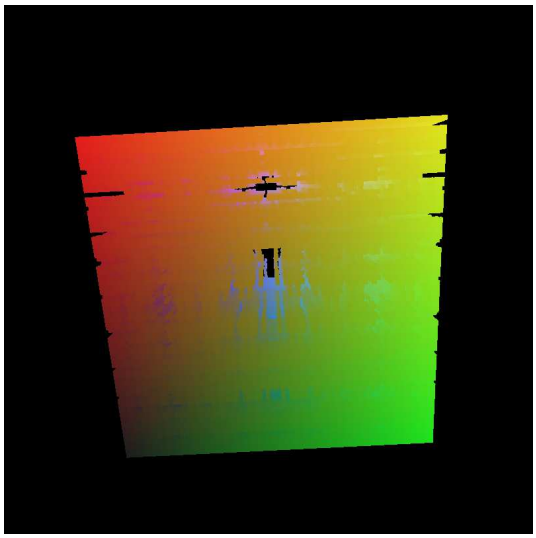
petit modèle

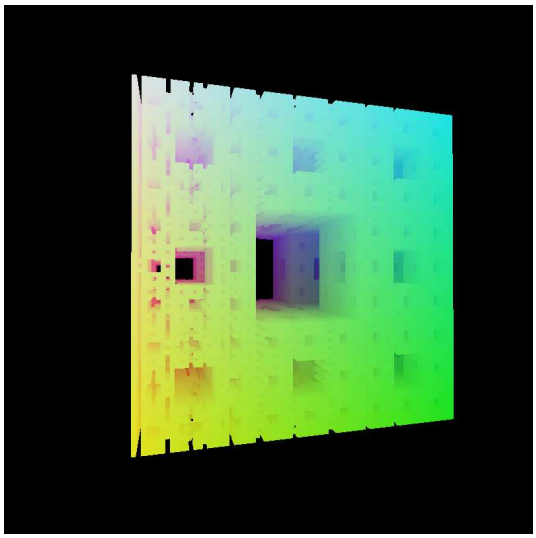




nombre de voxels communs avec le modèle : 18851 (92.3%)

grand modèle

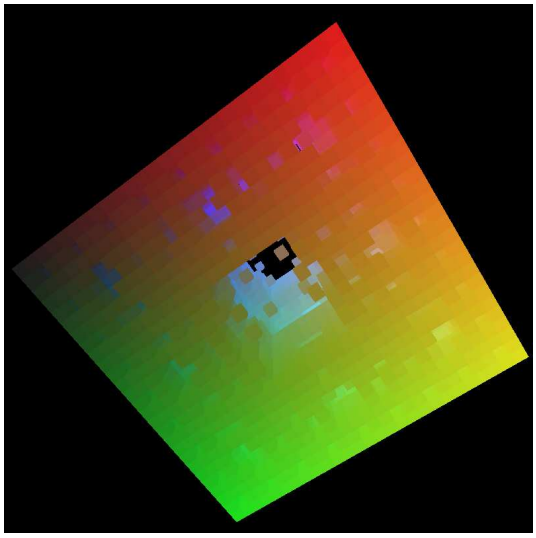


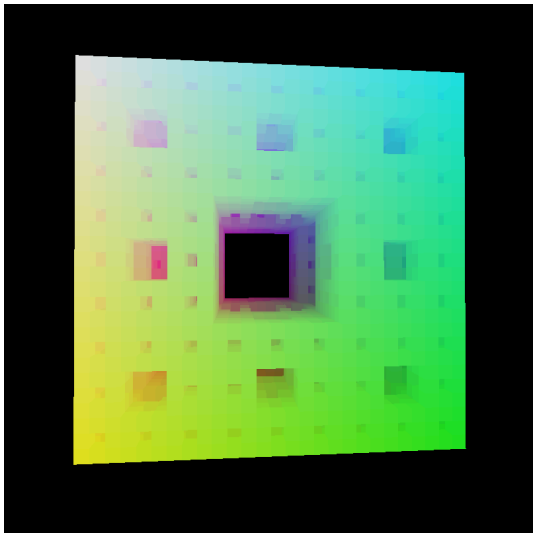


nombre de voxels communs avec le modèle : 513041 (91.2%)

E Les reconstructions pour 3 et 4 itérations avec deux projections.

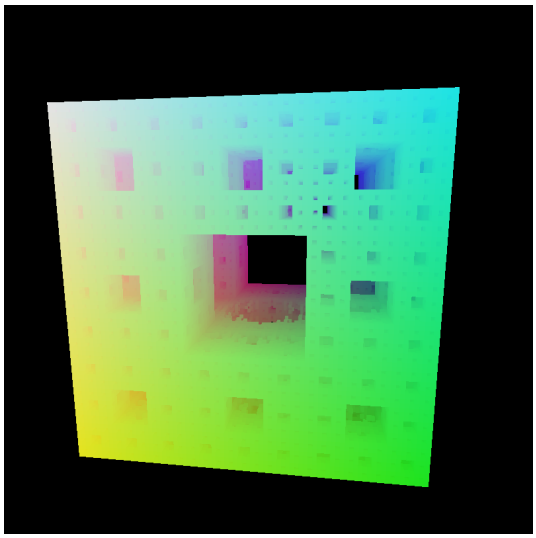
petit modèle

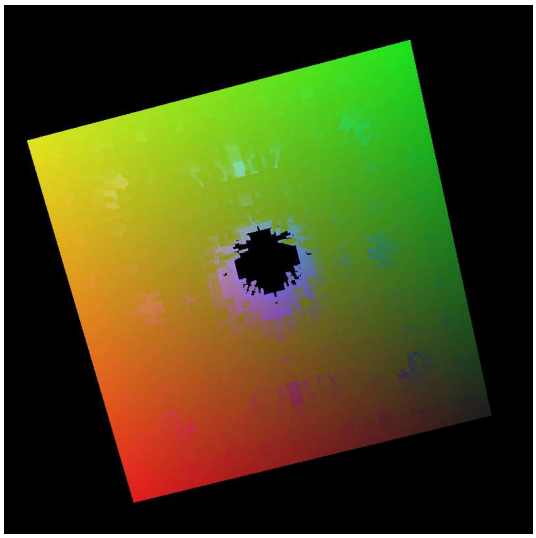




nombre de voxels communs avec le modèle : 18167 (95.8%)

grand modèle





nombre de voxels communs avec le modèle : 484497 (96.5%)