Plantando anomalias de densidade com gradiometria gravimétrica

Leonardo Uieda

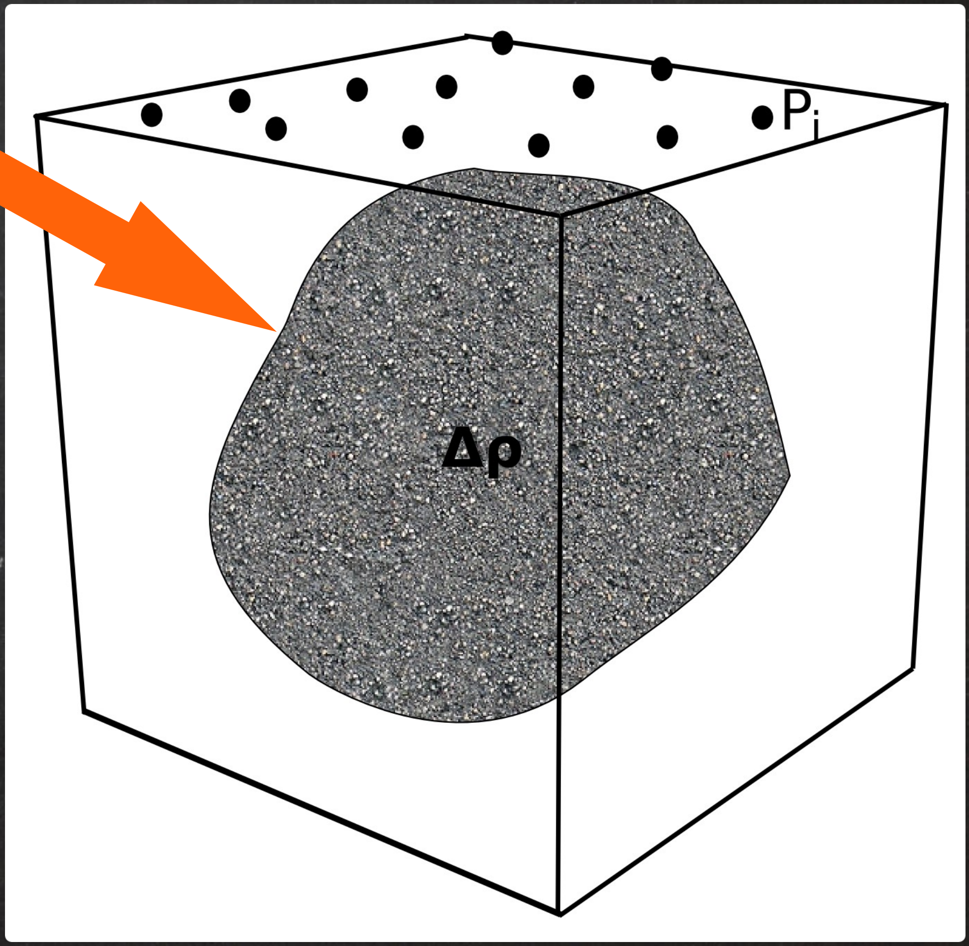
Observatório Nacional

2010

Modelagem Direta

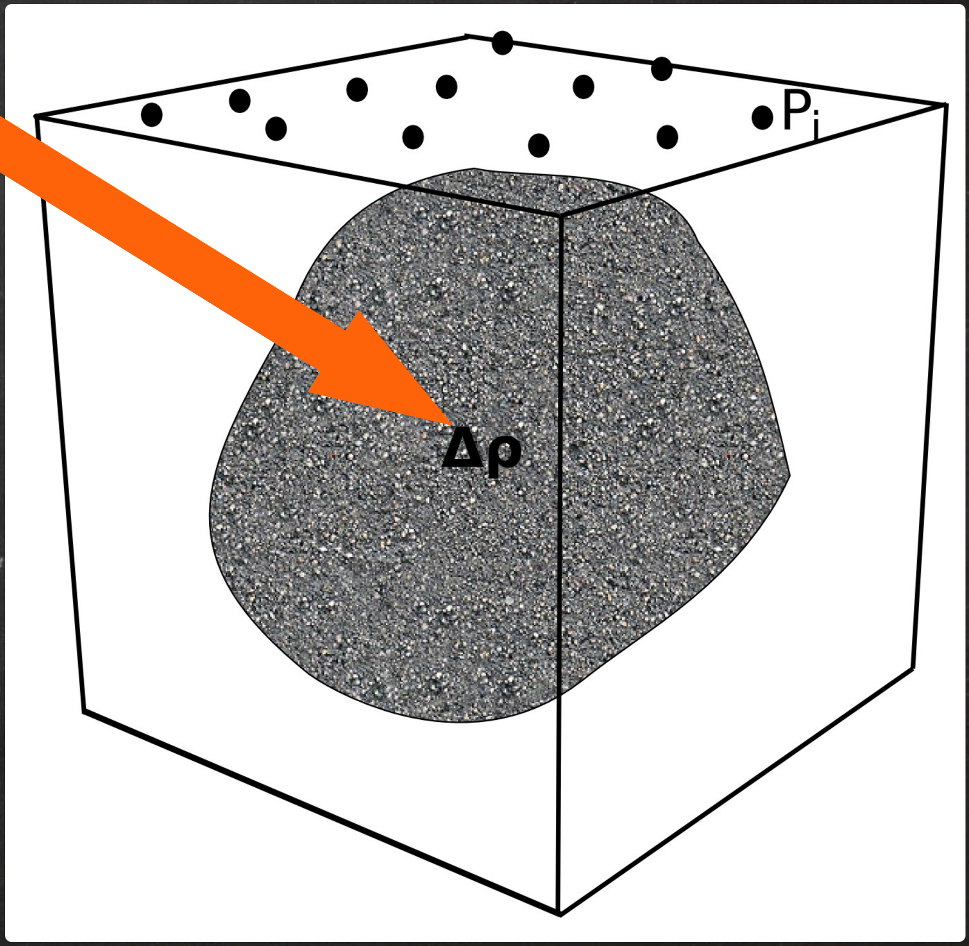
Modelagem direta

- Corpo anômalo



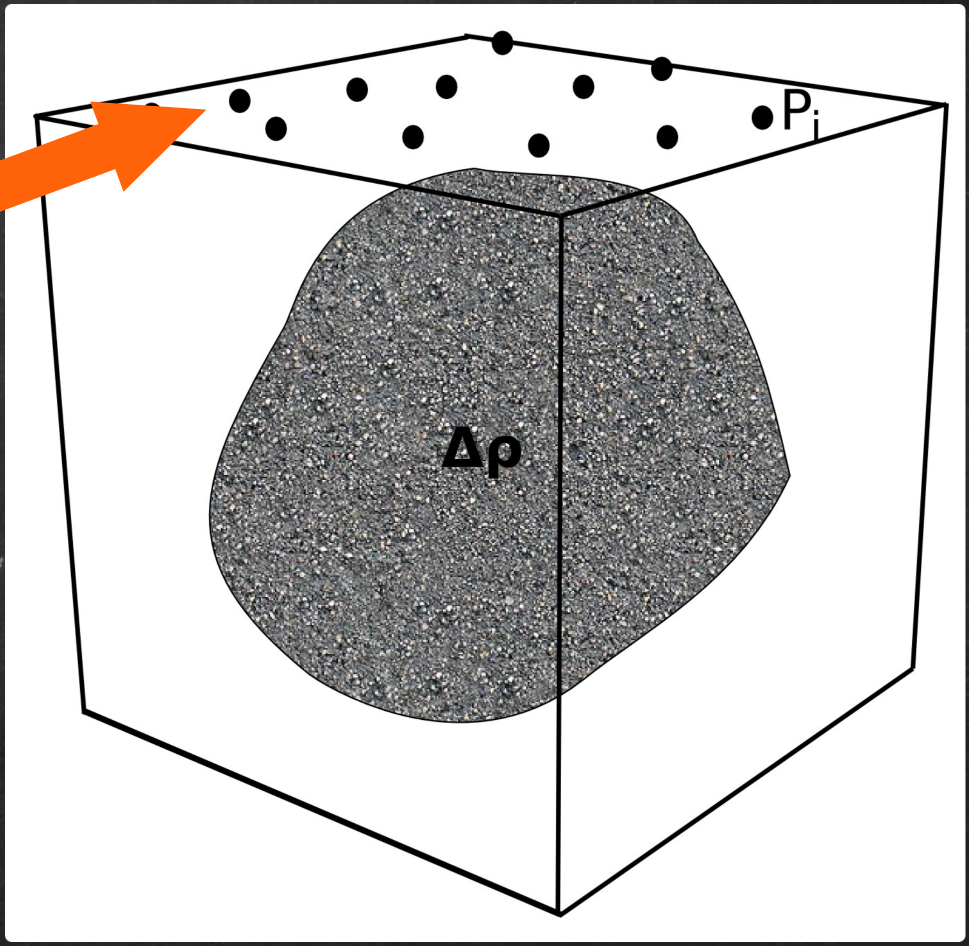
Modelagem direta

- ✔ Corpo anômalo
- ✔ Contraste de densidade



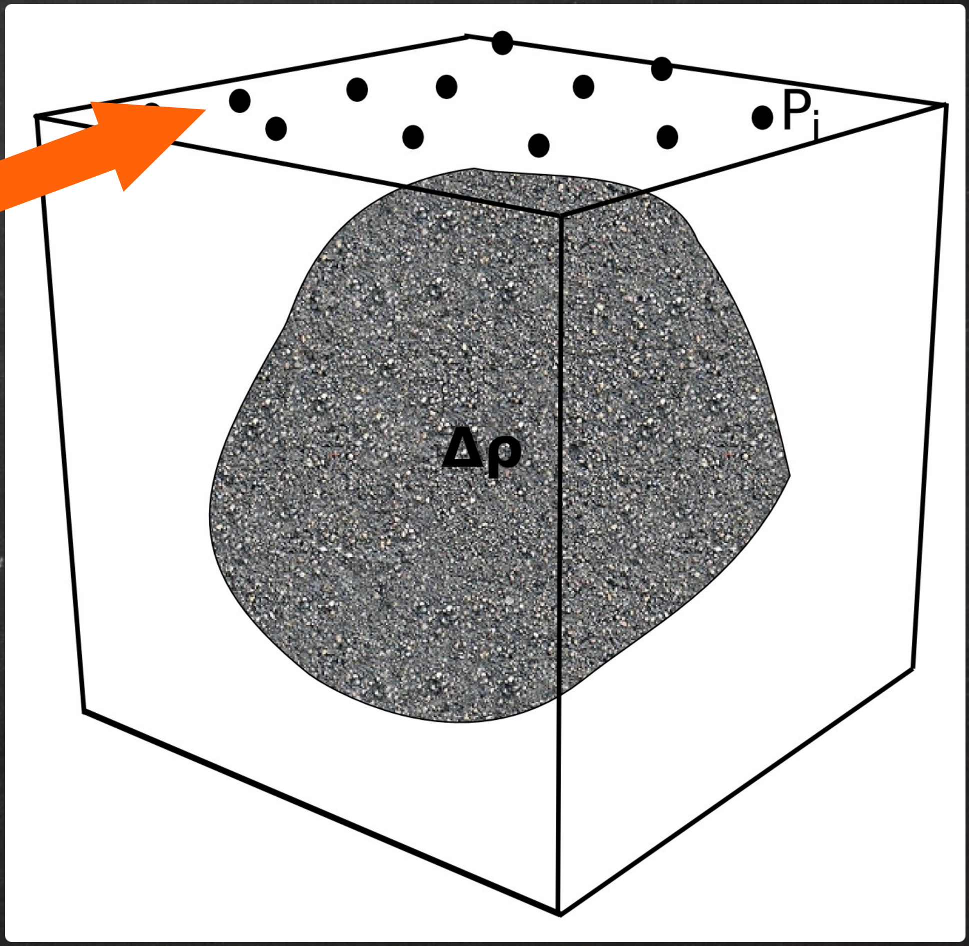
Modelagem direta

- Corpo anômalo
- Contraste de densidade
- Medidas em pontos



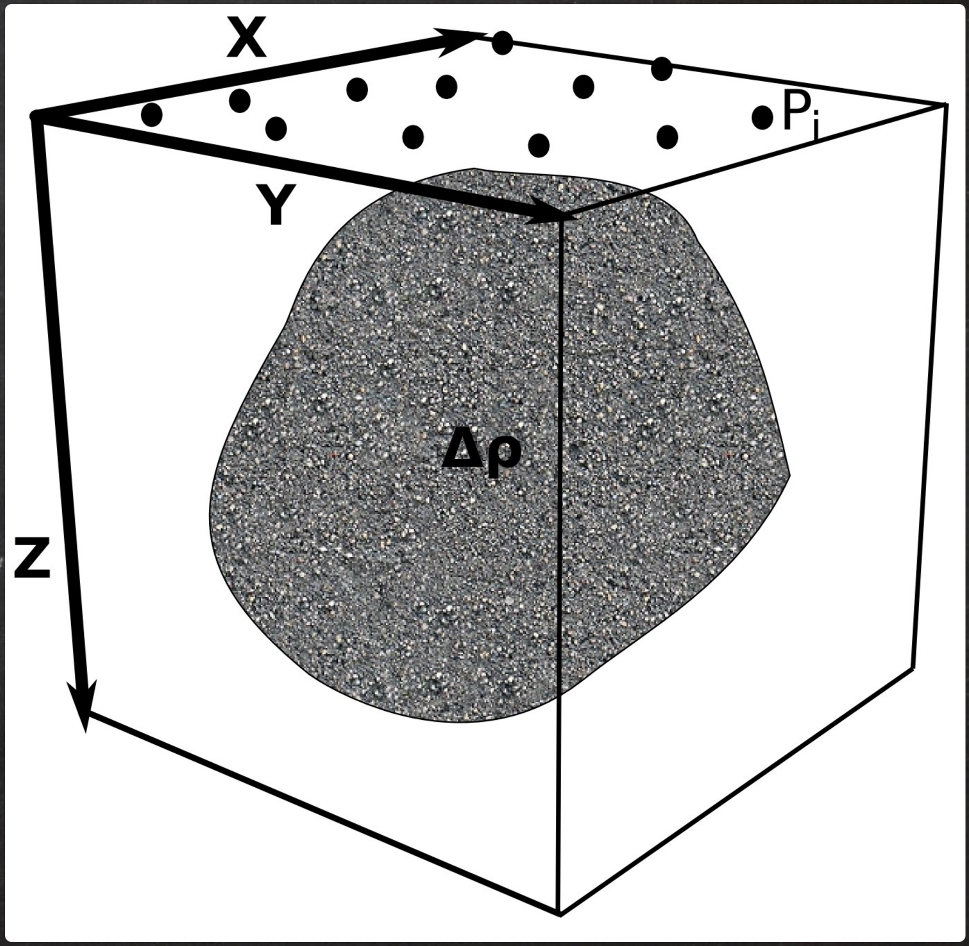
Modelagem direta

- ✔ Corpo anômalo
- ✔ Contraste de densidade
- ✔ Medidas em pontos
  - ✔ Gradientes de gravidade
  - ✔ 6 componentes



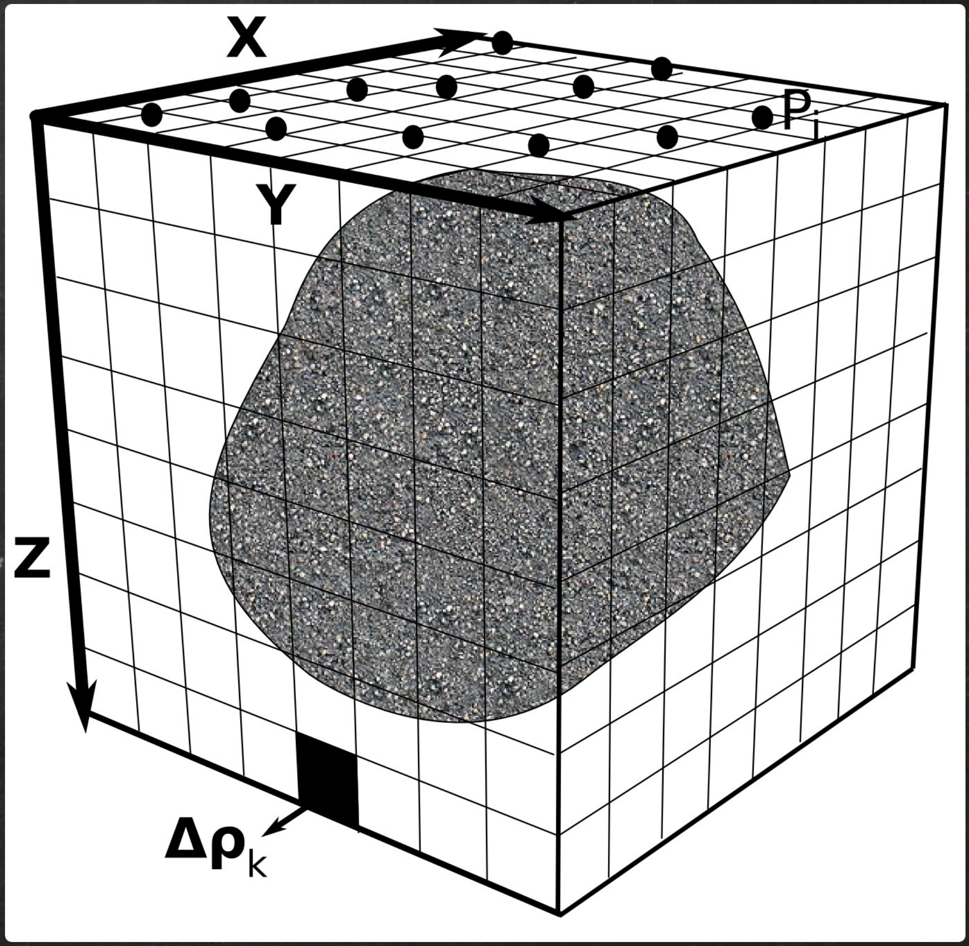
Modelagem direta

- ✔ Corpo anômalo
- ✔ Contraste de densidade
- ✔ Medidas em pontos
  - ✔ Gradientes de gravidade
  - ✔ 6 componentes
- ✔ Sistema de coordenadas



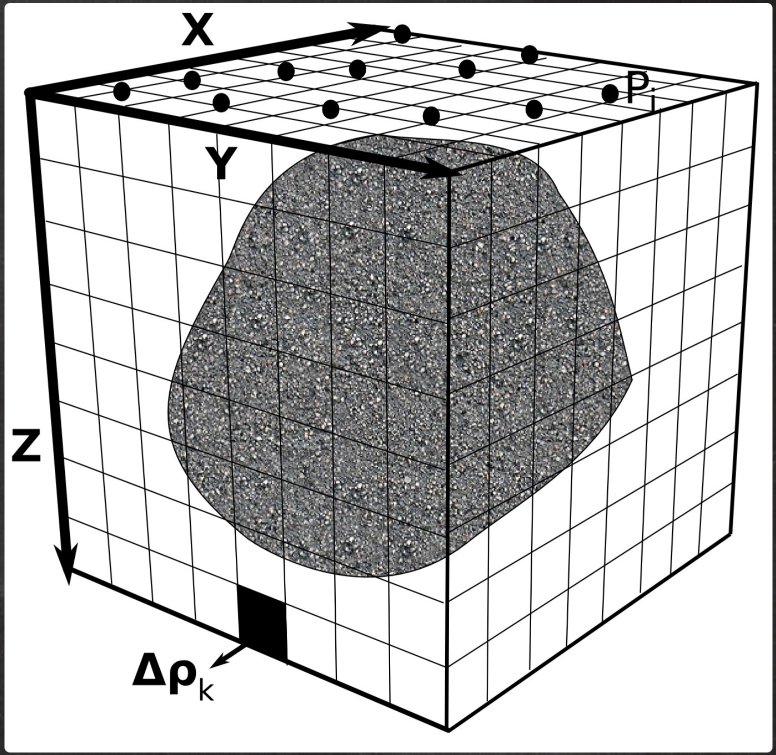
Modelagem direta

- ✔ Corpo anômalo
- ✔ Contraste de densidade
- ✔ Medidas em pontos
  - ✔ Gradientes de gravidade
  - ✔ 6 componentes
- ✔ Sistema de coordenadas
- ✔ Discretizar em prismas



Modelagem direta

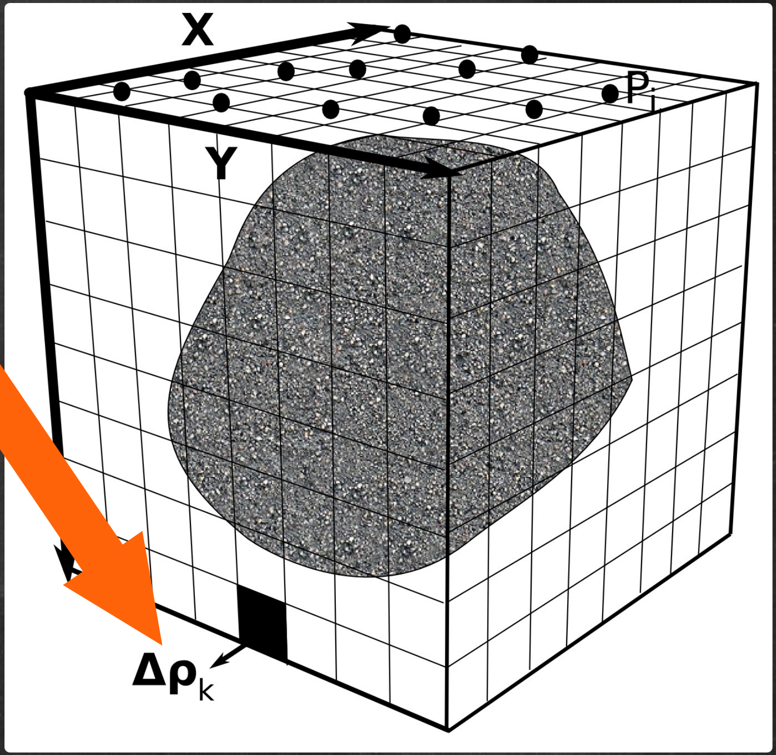
Cada prisma:



Modelagem direta

Cada prisma:

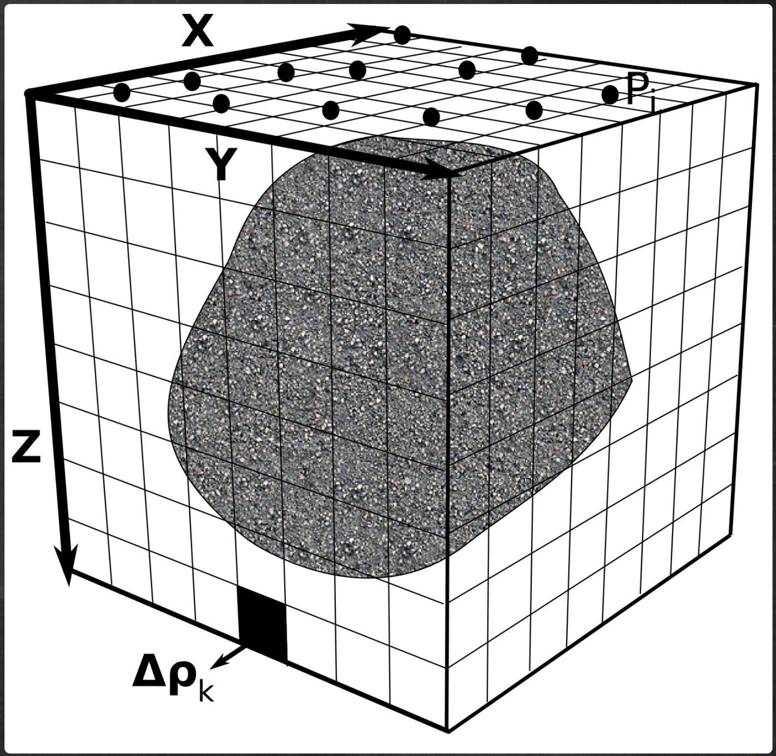
- ✔ Contraste de densidade $\Delta\rho_k$



Modelagem direta

Cada prisma:

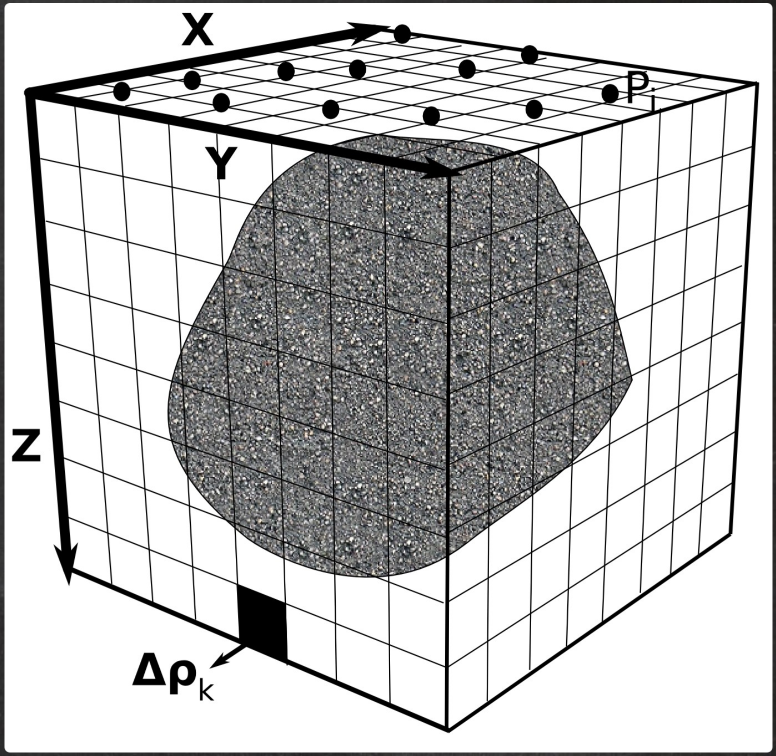
- ✔ Contraste de densidade $\Delta \rho_k$
- ✔ Volume V_k



Modelagem direta

Cada prisma:

- ✔ Contraste de densidade $\Delta\rho_k$
- ✔ Volume V_k



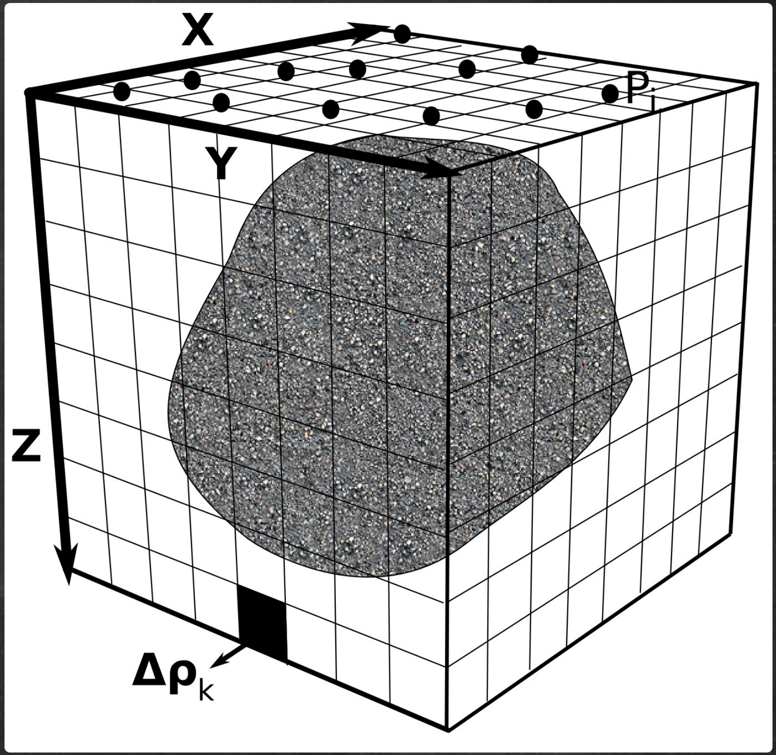
Componente α,β do gradiente:

$$g_{\alpha,\beta}(P_i)=\gamma\sum_{k=0}^{M-1}\Delta\rho_k\int_{V_k}K_{\alpha,\beta}\,dV$$

Modelagem direta

Cada prisma:

- ✔ Contraste de densidade $\Delta \rho_k$
- ✔ Volume V_k



Componente α,β do gradiente:

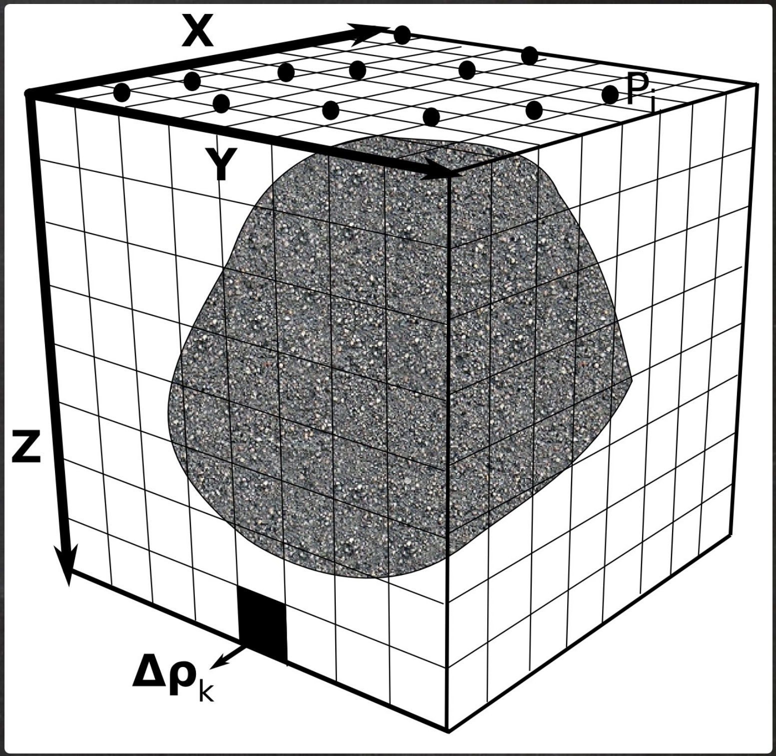
$$g_{\alpha,\beta}(P_i) = \gamma \sum_{k=0}^{M-1} \Delta \rho_k \int_{V_k} K_{\alpha,\beta}\, dV$$

Medido

Modelagem direta

Cada prisma:

- ✔ Contraste de densidade $\Delta\rho_k$
- ✔ Volume V_k



Componente α,β do gradiente:

$$g_{\alpha,\beta}(P_i) = \gamma \sum_{k=0}^{M-1} \Delta\rho_k \int_{V_k} K_{\alpha,\beta}\, dV$$

Medido

Quero saber

Modelagem Inversa

Modelagem inversa

Medições em diversos pontos:

$$g_{xx}(P_1) = \Delta\rho_1 \gamma \int_{V_1} K_{xx}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{xx}\, dV$$

$$\vdots$$

$$g_{xx}(P_{N_{xx}}) = \Delta\rho_1 \gamma \int_{V_1} K_{xx}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{xx}\, dV$$

$$\vdots$$

$$g_{zz}(P_{N_{zz}}) = \Delta\rho_1 \gamma \int_{V_1} K_{zz}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{zz}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{zz}\, dV$$

Modelagem inversa

Medições em diversos pontos:

$$\boxed{g_{xx}(P_1)} = \Delta\rho_1 \gamma \int_{V_1} K_{xx}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{xx}\, dV$$

$$\vdots$$

$$\boxed{g_{xx}(P_{N_{xx}})} = \Delta\rho_1 \gamma \int_{V_1} K_{xx}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{xx}\, dV$$

$$\vdots$$

$$\boxed{g_{zz}(P_{N_{zz}})} = \Delta\rho_1 \gamma \int_{V_1} K_{zz}\, dV + \Delta\rho_2 \gamma \int_{V_2} K_{zz}\, dV + \cdots + \Delta\rho_M \gamma \int_{V_M} K_{zz}\, dV$$

Modelagem inversa

Medições em diversos pontos:

$$\boxed{g_{xx}(P_1)} = \boxed{\Delta\rho_1 \gamma \int_{V_1} K_{xx}\,dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\,dV + \cdots + \Delta\rho_M} \gamma \int_{V_M} K_{xx}\,dV$$

$$\vdots$$

$$\boxed{g_{xx}(P_{N_{xx}})} = \boxed{\Delta\rho_1 \gamma \int_{V_1} K_{xx}\,dV + \Delta\rho_2 \gamma \int_{V_2} K_{xx}\,dV + \cdots + \Delta\rho_M} \gamma \int_{V_M} K_{xx}\,dV$$

$$\vdots$$

$$\boxed{g_{zz}(P_{N_{zz}})} = \boxed{\Delta\rho_1 \gamma \int_{V_1} K_{zz}\,dV + \Delta\rho_2 \gamma \int_{V_2} K_{zz}\,dV + \cdots + \Delta\rho_M} \gamma \int_{V_M} K_{zz}\,dV$$

Modelagem inversa

Medições em diversos pontos:

$$
\begin{bmatrix} g_{xx}(P_1) \\ \vdots \\ g_{xx}(P_{N_{xx}}) \\ \vdots \\ \vdots \\ g_{zz}(P_{N_{zz}}) \end{bmatrix} = \begin{bmatrix} \gamma \int_{V_1} K_{xx}\, dV & \gamma \int_{V_2} K_{xx}\, dV & \cdots & \gamma \int_{V_M} K_{xx}\, dV \\ & \vdots & & \\ \gamma \int_{V_1} K_{xx}\, dV & \gamma \int_{V_2} K_{xx}\, dV & \cdots & \gamma \int_{V_M} K_{xx}\, dV \\ & \vdots & & \\ & \vdots & & \\ \gamma \int_{V_1} K_{zz}\, dV & \gamma \int_{V_2} K_{zz}\, dV & \cdots & \gamma \int_{V_M} K_{zz}\, dV \end{bmatrix} \begin{bmatrix} \Delta\rho_1 \\ \vdots \\ \Delta\rho_N \\ \vdots \\ \vdots \\ \Delta\rho_M \end{bmatrix}
$$

Modelagem inversa

Medições em diversos pontos:

$$\boxed{\begin{bmatrix} g_{xx}(P_1) \\ \vdots \\ g_{xx}(P_{N_{xx}}) \\ \vdots \\ \vdots \\ g_{zz}(P_{N_{zz}}) \end{bmatrix}} = \begin{bmatrix} \gamma \int\limits_{V_1} K_{xx} dV & \gamma \int\limits_{V_2} K_{xx} dV & \cdots & \gamma \int\limits_{V_M} K_{xx} dV \\ & \vdots & & \\ \gamma \int\limits_{V_1} K_{xx} dV & \gamma \int\limits_{V_2} K_{xx} dV & \cdots & \gamma \int\limits_{V_M} K_{xx} dV \\ & \vdots & & \\ & \vdots & & \\ \gamma \int\limits_{V_1} K_{zz} dV & \gamma \int\limits_{V_2} K_{zz} dV & \cdots & \gamma \int\limits_{V_M} K_{zz} dV \end{bmatrix} \boxed{\begin{bmatrix} \Delta\rho_1 \\ \vdots \\ \Delta\rho_N \\ \vdots \\ \vdots \\ \Delta\rho_M \end{bmatrix}}$$

Modelagem inversa

Medições em diversos pontos:

$$
\underbrace{\begin{bmatrix} g_{xx}(P_1) \\ \vdots \\ g_{xx}(P_{N_{xx}}) \\ \vdots \\ g_{zz}(P_{N_{zz}}) \end{bmatrix}}_{\bar{d}_0 =}
=
\underbrace{\begin{bmatrix}
\gamma \int_{V_1} K_{xx}\, dV & \gamma \int_{V_2} K_{xx}\, dV & \cdots & \gamma \int_{V_M} K_{xx}\, dV \\
 & \vdots & & \\
\gamma \int_{V_1} K_{xx}\, dV & \gamma \int_{V_2} K_{xx}\, dV & \cdots & \gamma \int_{V_M} K_{xx}\, dV \\
 & \vdots & & \\
\gamma \int_{V_1} K_{zz}\, dV & \gamma \int_{V_2} K_{zz}\, dV & \cdots & \gamma \int_{V_M} K_{zz}\, dV
\end{bmatrix}}_{\bar{\bar{G}}}
\underbrace{\begin{bmatrix} \Delta\rho_1 \\ \vdots \\ \Delta\rho_N \\ \vdots \\ \Delta\rho_M \end{bmatrix}}_{\bar{p}}
$$

Modelagem inversa

Medições em diversos pontos:

$$\bar{d}_0 = \begin{bmatrix} g_{xx}(P_1) \\ \vdots \\ g_{xx}(P_{N_{xx}}) \\ \vdots \\ \vdots \\ g_{zz}(P_{N_{zz}}) \end{bmatrix} = \begin{bmatrix} \gamma \int_{V_1} K_{xx} dV & \gamma \int_{V_2} K_{xx} dV & \cdots & \gamma \int_{V_M} K_{xx} dV \\ & \vdots & & \\ \gamma \int_{V_1} K_{xx} dV & \gamma \int_{V_2} K_{xx} dV & \cdots & \gamma \int_{V_M} K_{xx} dV \\ & \vdots & & \\ \gamma \int_{V_1} K_{zz} dV & \gamma \int_{V_2} K_{zz} dV & \cdots & \gamma \int_{V_M} K_{zz} dV \end{bmatrix} \begin{bmatrix} \Delta\rho_1 \\ \vdots \\ \Delta\rho_N \\ \vdots \\ \Delta\rho_M \end{bmatrix}$$

$\bar{\bar{G}}$ ➡ Jacobiana $\quad \bar{p}$

Modelagem inversa

Medições em diversos pontos:

$$\begin{bmatrix} g_{xx}(P_1) \\ \vdots \\ g_{xx}(P_{N_{xx}}) \\ \vdots \\ \vdots \\ g_{zz}(P_{N_{zz}}) \end{bmatrix} = \begin{bmatrix} \gamma \int_{V_1} K_{xx} dV & \gamma \int_{V_2} K_{xx} dV & \cdots & \gamma \int_{V_M} K_{xx} dV \\ & \vdots & & \\ \gamma \int_{V_1} K_{xx} dV & \gamma \int_{V_2} K_{xx} dV & \cdots & \gamma \int_{V_M} K_{xx} dV \\ & \vdots & & \\ \gamma \int_{V_1} K_{zz} dV & \gamma \int_{V_2} K_{zz} dV & \cdots & \gamma \int_{V_M} K_{zz} dV \end{bmatrix} \begin{bmatrix} \Delta\rho_1 \\ \vdots \\ \Delta\rho_N \\ \vdots \\ \Delta\rho_M \end{bmatrix}$$

$\bar{d}_0 =$

$\bar{\bar{G}} \Rightarrow$ Jacobiana

$\bar{p}$

Dados preditos pelos parâmetros

Modelagem inversa

✔ Vetor de dados medidos: $\bar{d}$

Modelagem inversa

- ✔ Vetor de dados medidos: $\bar{d}$
- ✔ Resíduos: $\bar{r} = \bar{d} - \bar{d}_0$

Modelagem inversa

- Vetor de dados medidos: $\bar{d}$
- Resíduos: $\bar{r} = \bar{d} - \bar{d}_0$

Dados preditos pelos parâmetros

Modelagem inversa

- ✔ Vetor de dados medidos: $\bar{d}$
- ✔ Resíduos: $\bar{r} = \bar{d} - \bar{d}_0$

Dados preditos pelos parâmetros

$$\bar{r} = \bar{d} - \bar{\bar{G}}\,\bar{p}$$

Modelagem inversa

- ✔ Vetor de dados medidos: $\bar{d}$
- ✔ Resíduos: $\bar{r} = \bar{d} - \bar{d}_0$

Dados preditos pelos parâmetros

$$\bar{r} = \bar{d} - \bar{\bar{G}}\,\bar{p}$$

Qual valor dos parâmetros que minimizam uma determinada norma dos resíduos?

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu \theta$$

Modelagem inversa

- Minimizar uma função objetivo
  - ϕ = Norma dos resíduos

$$\Gamma = \phi + \mu \theta$$

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu\theta$$

- ✔ ϕ = Norma dos resíduos
- ✔ θ = Regularizadora (vínculo)

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu\theta$$

- ✔ ϕ = Norma dos resíduos
- ✔ θ = Regularizadora (vínculo)
- ✔ μ = Parâmetro de regularização

Modelagem inversa

- Minimizar uma função objetivo

$$\Gamma = \phi + \mu \theta$$

  - ϕ = Norma dos resíduos
  - θ = Regularizadora (vínculo)
  - μ = Parâmetro de regularização

- Solução analítica

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu\theta$$

  - ✔ ϕ = Norma dos resíduos
  - ✔ θ = Regularizadora (vínculo)
  - ✔ μ = Parâmetro de regularização

- ✔ Solução analítica
- ✔ Busca aleatória (métodos heurísticos)

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu\theta$$

  - ✔ ϕ = Norma dos resíduos
  - ✔ θ = Regularizadora (vínculo)
  - ✔ μ = Parâmetro de regularização

- ✔ Solução analítica
- ✔ Busca aleatória (métodos heurísticos)
- ✔ Busca sistemática

Modelagem inversa

- ✔ Minimizar uma função objetivo

$$\Gamma = \phi + \mu\theta$$

  - ✔ ϕ = Norma dos resíduos
  - ✔ θ = Regularizadora (vínculo)
  - ✔ μ = Parâmetro de regularização

- ✔ Solução analítica
- ✔ Busca aleatória (métodos heurísticos)
- ✔ Busca sistemática ← Utilizada neste trabalho

Modelagem inversa

- Solução analítica $\hat{\bar{p}} = (\bar{\bar{G}}^T \bar{\bar{G}} + \mu \bar{\bar{W}})^{-1} \bar{\bar{G}}^T \bar{d}$
  - Resolver um sistema NxN ou MxM

Modelagem inversa

- ✔ Solução analítica $\hat{\bar{p}} = (\bar{\bar{G}}^T \bar{\bar{G}} + \mu \bar{\bar{W}})^{-1} \bar{\bar{G}}^T \bar{d}$
  - ✔ Resolver um sistema NxN ou MxM
- ✔ Busca aleatória (métodos heurísticos)
  - ✔ Testar diversos parâmetros aleatoriamente

Modelagem inversa

- Solução analítica $\hat{\bar{p}} = (\bar{\bar{G}}^T \bar{\bar{G}} + \mu \bar{\bar{W}})^{-1} \bar{\bar{G}}^T \bar{d}$
  - Resolver um sistema NxN ou MxM
- Busca aleatória (métodos heurísticos)
  - Testar diversos parâmetros aleatoriamente
- Busca sistemática
  - Testar diversos parâmetros sistematicamente
  - Rene (1986)
  - Camacho *et al.* (2000)

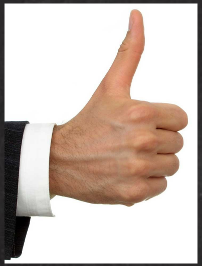
Modelagem inversa

- Solução analítica $\hat{\bar{p}} = (\bar{\bar{G}}^T \bar{\bar{G}} + \mu \bar{\bar{W}})^{-1} \bar{\bar{G}}^T \bar{d}$
  - Resolver um sistema NxN ou MxM
- Busca aleatória (métodos heurísticos)
  - Testar diversos parâmetros aleatoriamente
- Busca sistemática
  - Testar diversos parâmetros sistematicamente
  - Rene (1986)
  - Camacho *et al.* (2000)

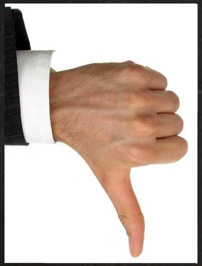
$\boxed{\text{Desenvolvido um híbrido}}$

Modelagem inversa

Rene (1986): Open-Reject-Fill



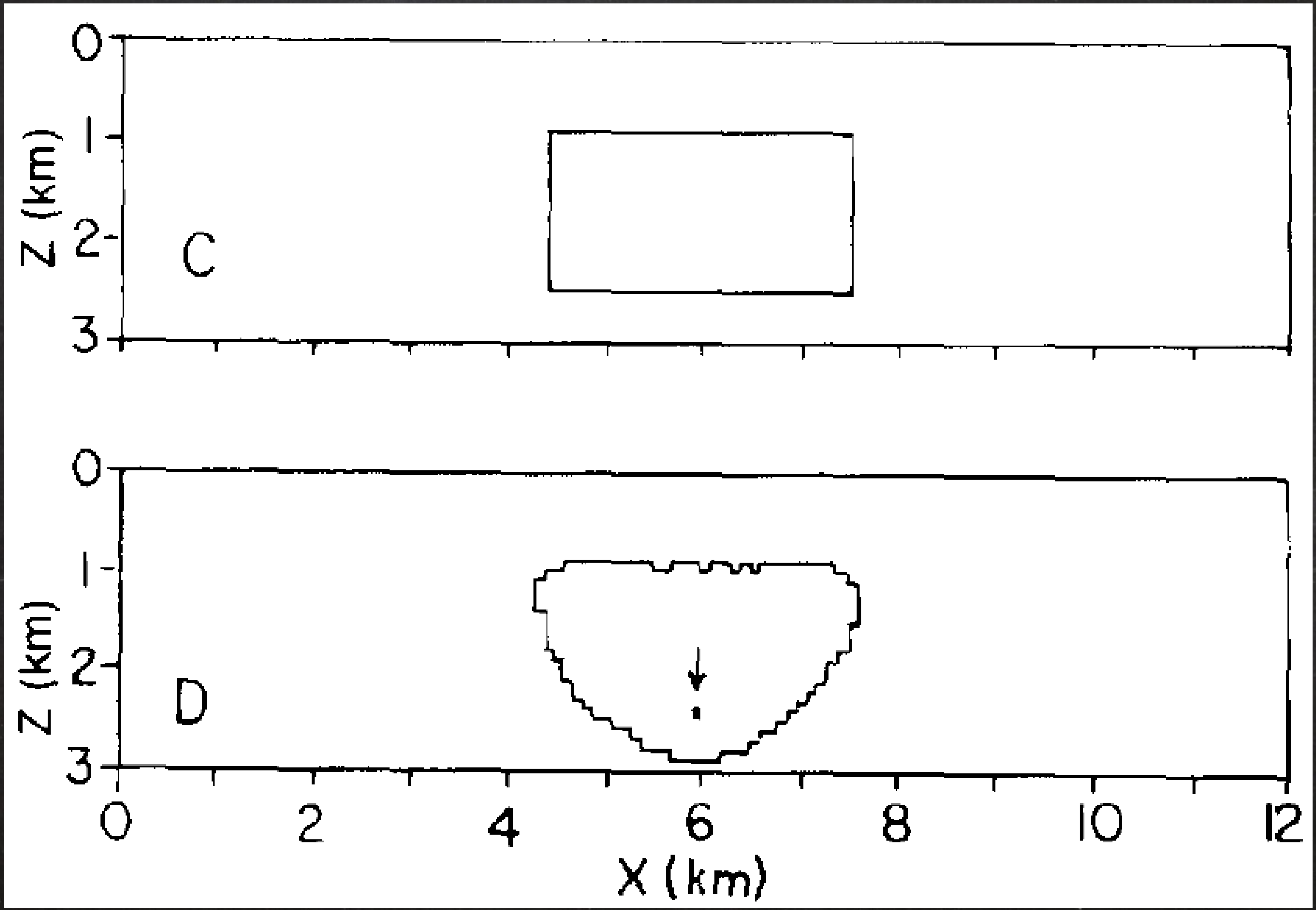
- ✔ Solução cresce entorno de "sementes"
- ✔ Busca restrita a "vizinhos"
- ✔ Solução compacta (limita vizinhos: "reject")



- ✗ Somente 1 contraste de densidade
- ✗ 2D

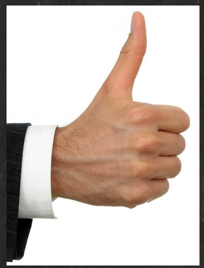
Modelagem inversa

Rene (1986): Open-Reject-Fill

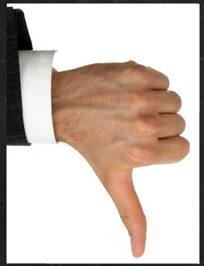


Modelagem inversa

Camacho *et al.* (2000): Growing bodies



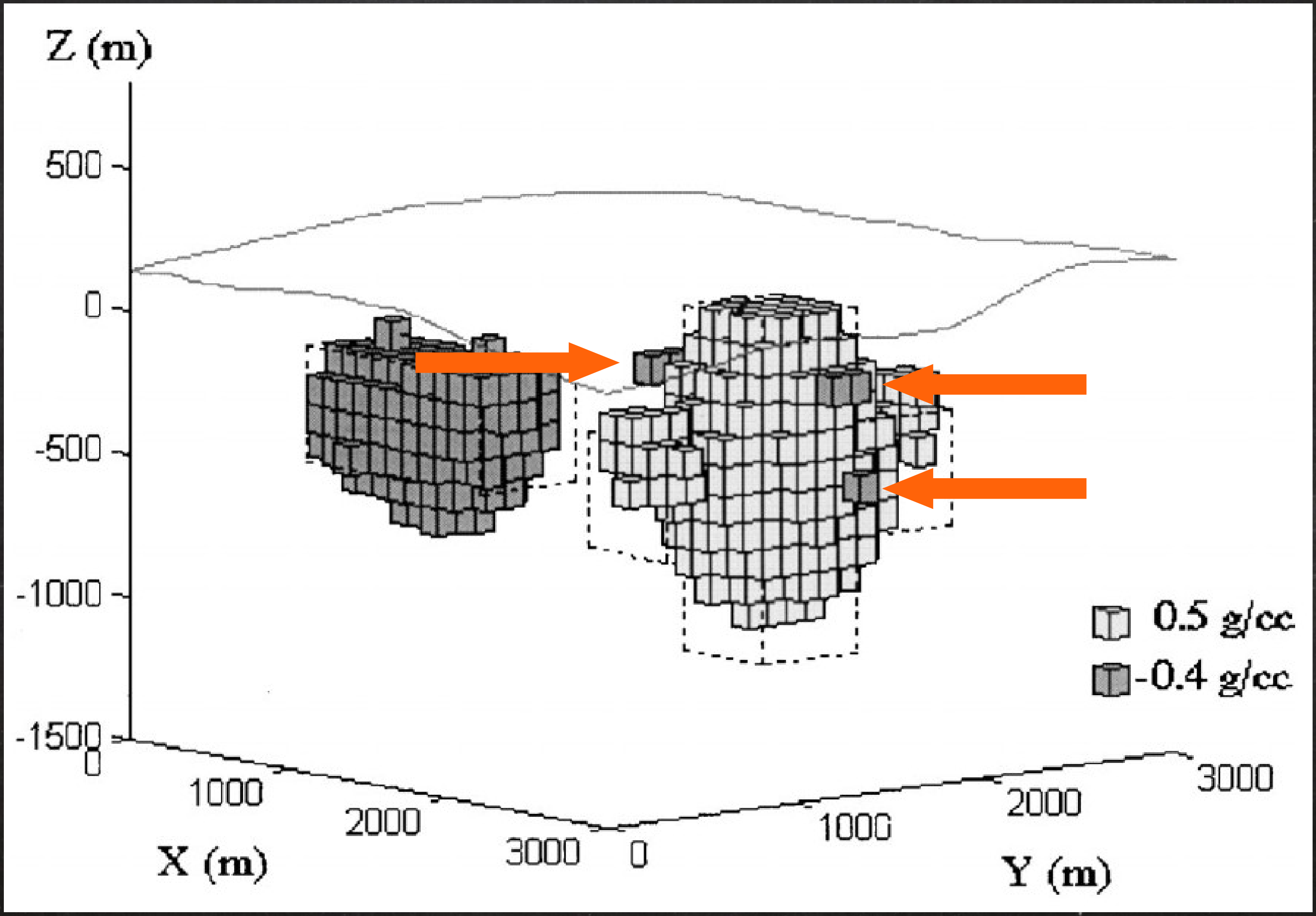
- ✔ 3D
- ✔ Não utiliza “sementes”
- ✔ Avalia todas células “vazias”



- ✗ Somente 2 contrastes de densidade
- ✗ Avalia todas células “vazias” = lento
- ✗ Não garante solução compacta

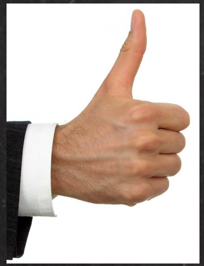
Modelagem inversa

Camacho *et al.* (2000): Growing bodies

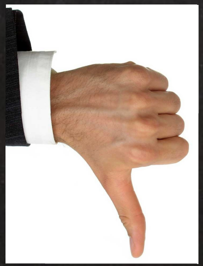


Modelagem inversa

Neste trabalho:



- ✔ 3D
- ✔ Utiliza "sementes"
- ✔ Contraste de densidade por "semente"
- ✔ Busca limitada a "vizinhos"
- ✔ Solução compacta (regularização)



- ✗ Fornecer sementes corretas
- ✔ Há métodos para isso (Beiki & Pedersen, 2010)

Modelagem inversa

Regularização:

$$\Gamma = \phi + \mu \theta$$

Função objetivo

$$\theta = \sum_{k=1}^{M} \frac{p_k}{\Delta \rho_{seed}} l_k^a$$

Regularizador

- ✔ l_k = distância do k-ésimo prisma a sua semente
- ✔ $\Delta \rho_{seed}$ = contraste de densidade da semente
- ✔ p_k = contraste de densidade do k-ésimo prisma
- ✔ a = potência

Modelagem inversa

Regularização:

$$\Gamma = \phi + \mu \theta \quad \Rightarrow \quad \theta = \sum_{k=1}^{M} \frac{p_k}{\Delta \rho_{seed}} l_k^a$$

Função objetivo

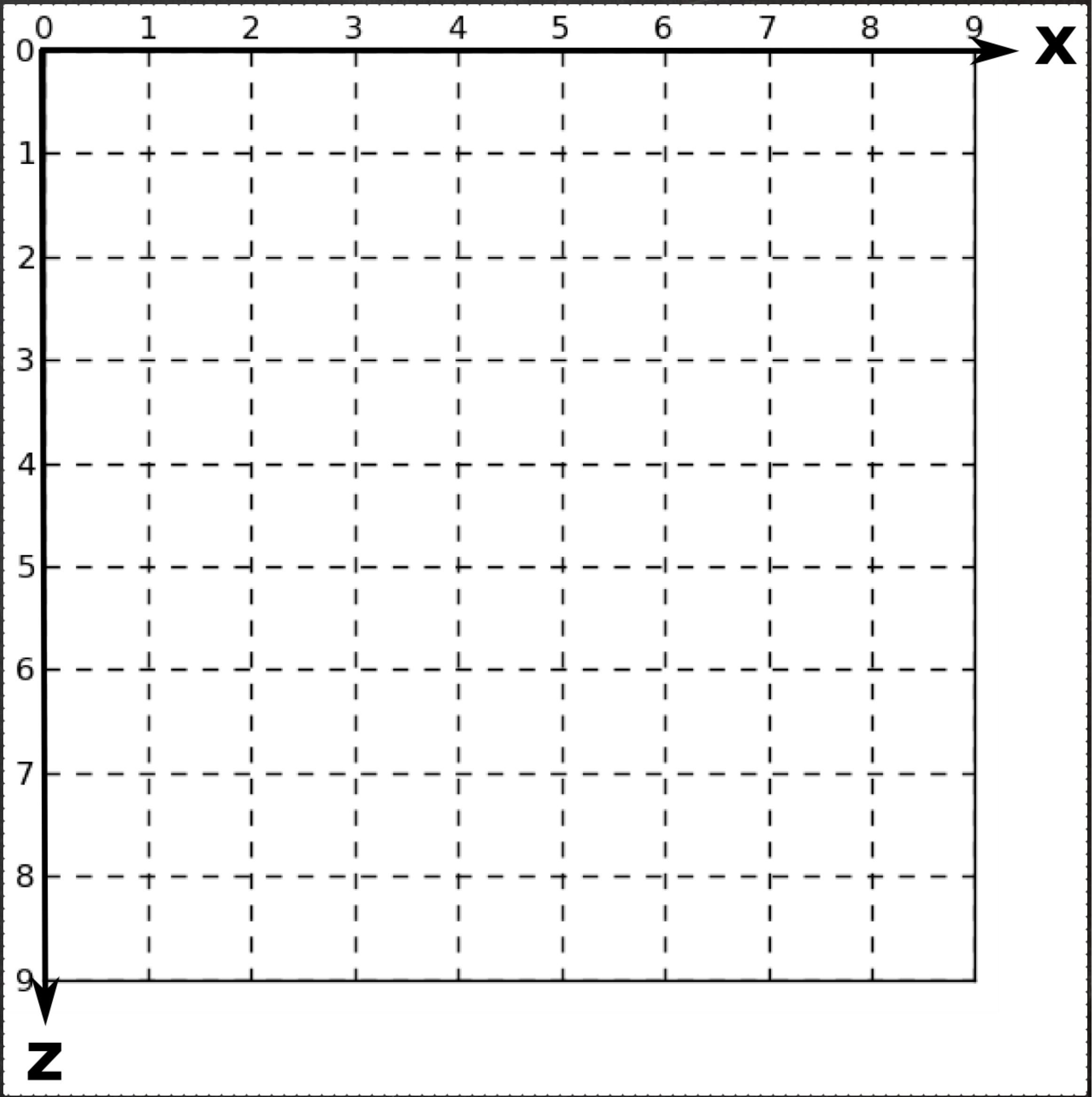
Regularizador

Minimizar θ impõe compacidade

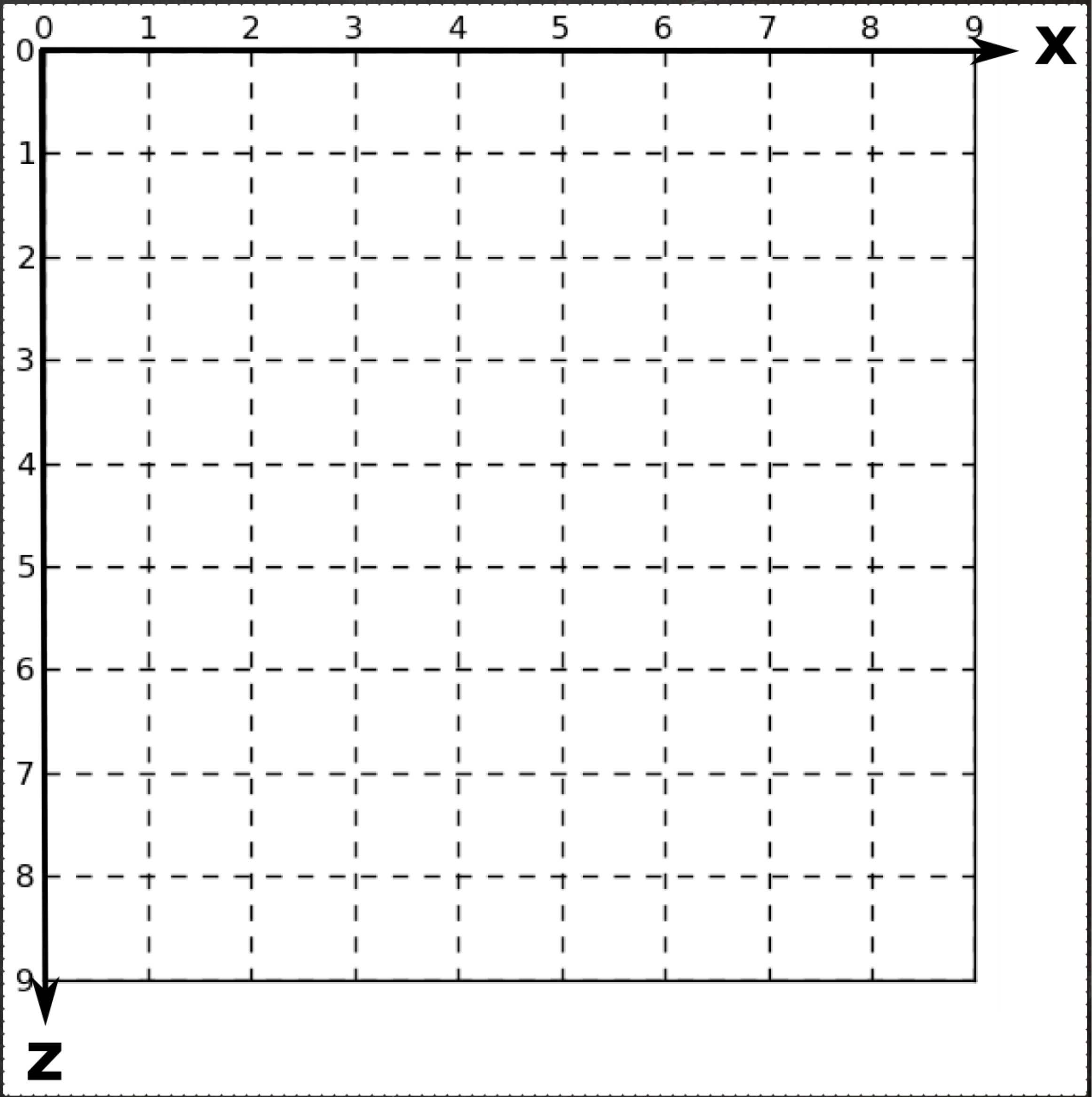
- ✔ l_k = distância do k-ésimo prisma a sua semente
- ✔ $\Delta \rho_{seed}$ = contraste de densidade da semente
- ✔ p_k = contraste de densidade do k-ésimo prisma
- ✔ a = potência

Algoritmo

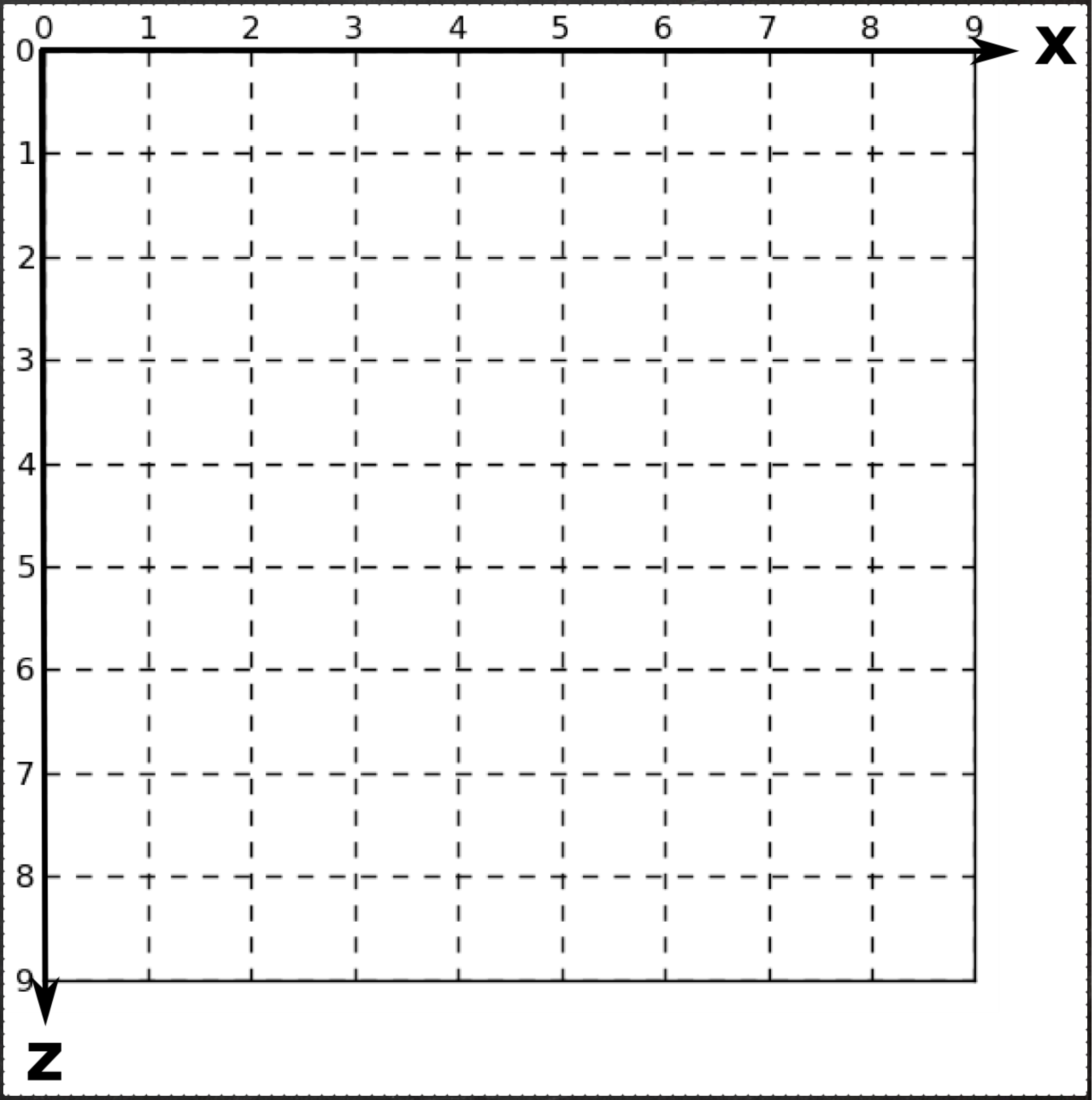
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



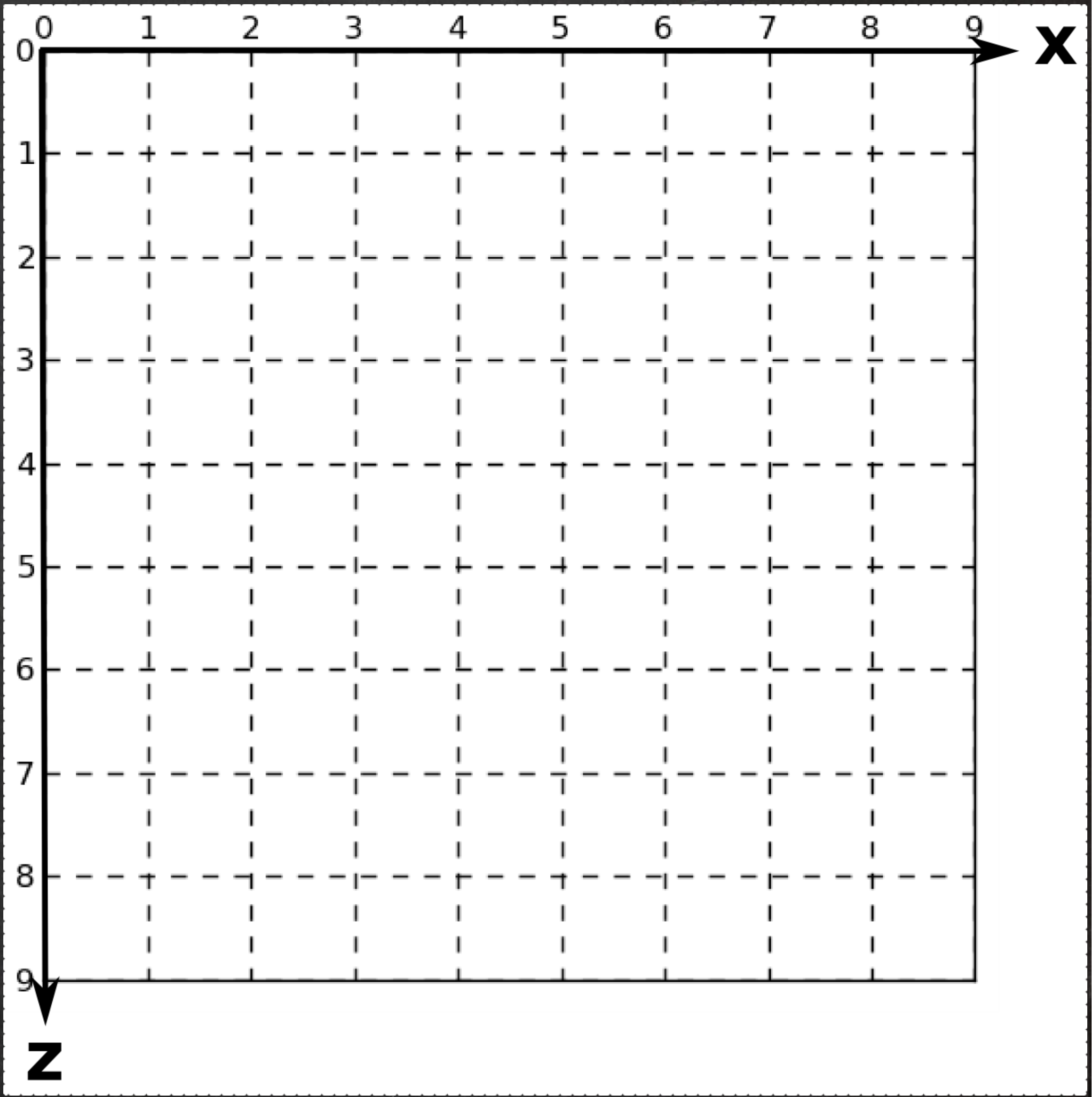
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



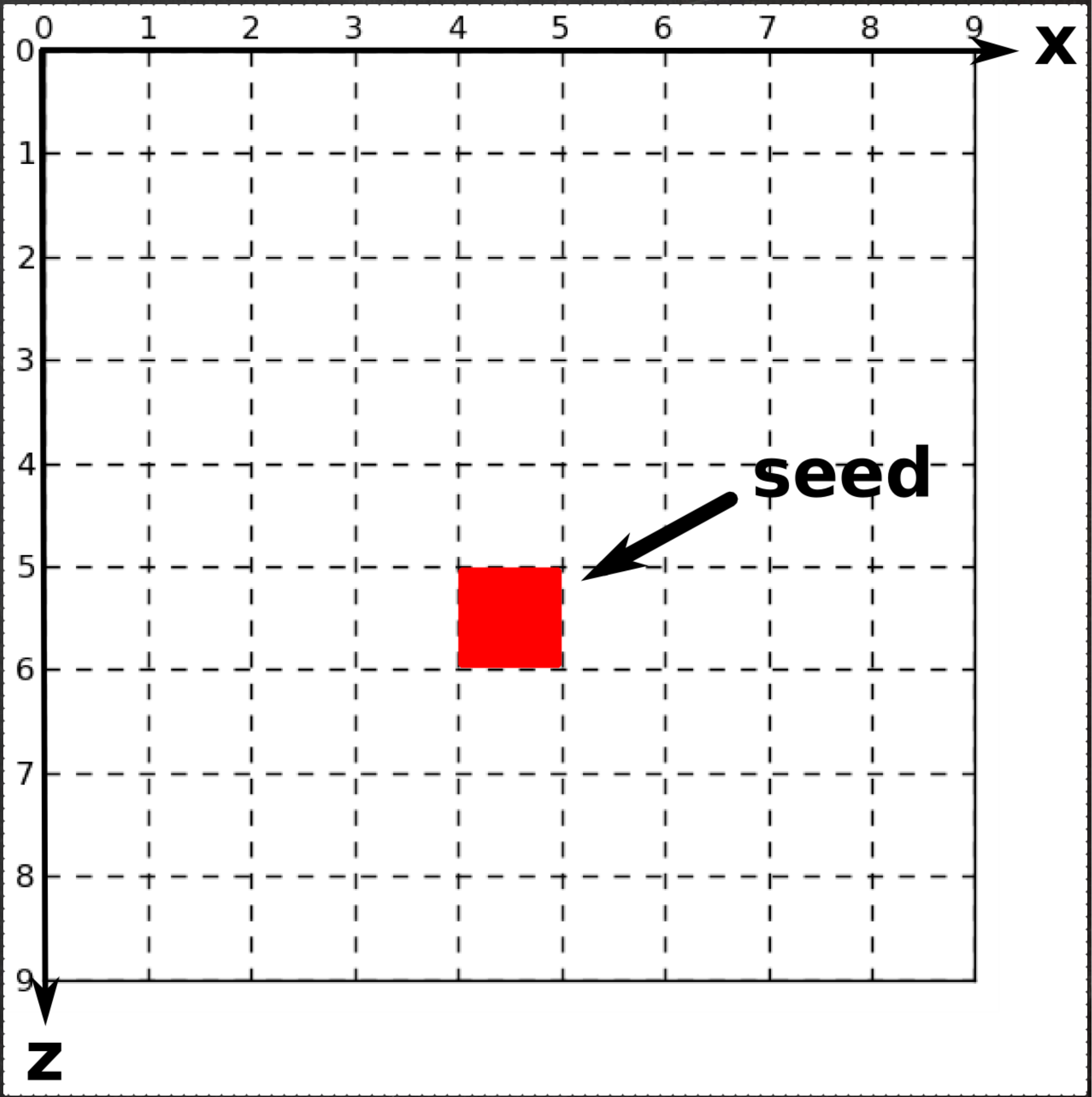
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



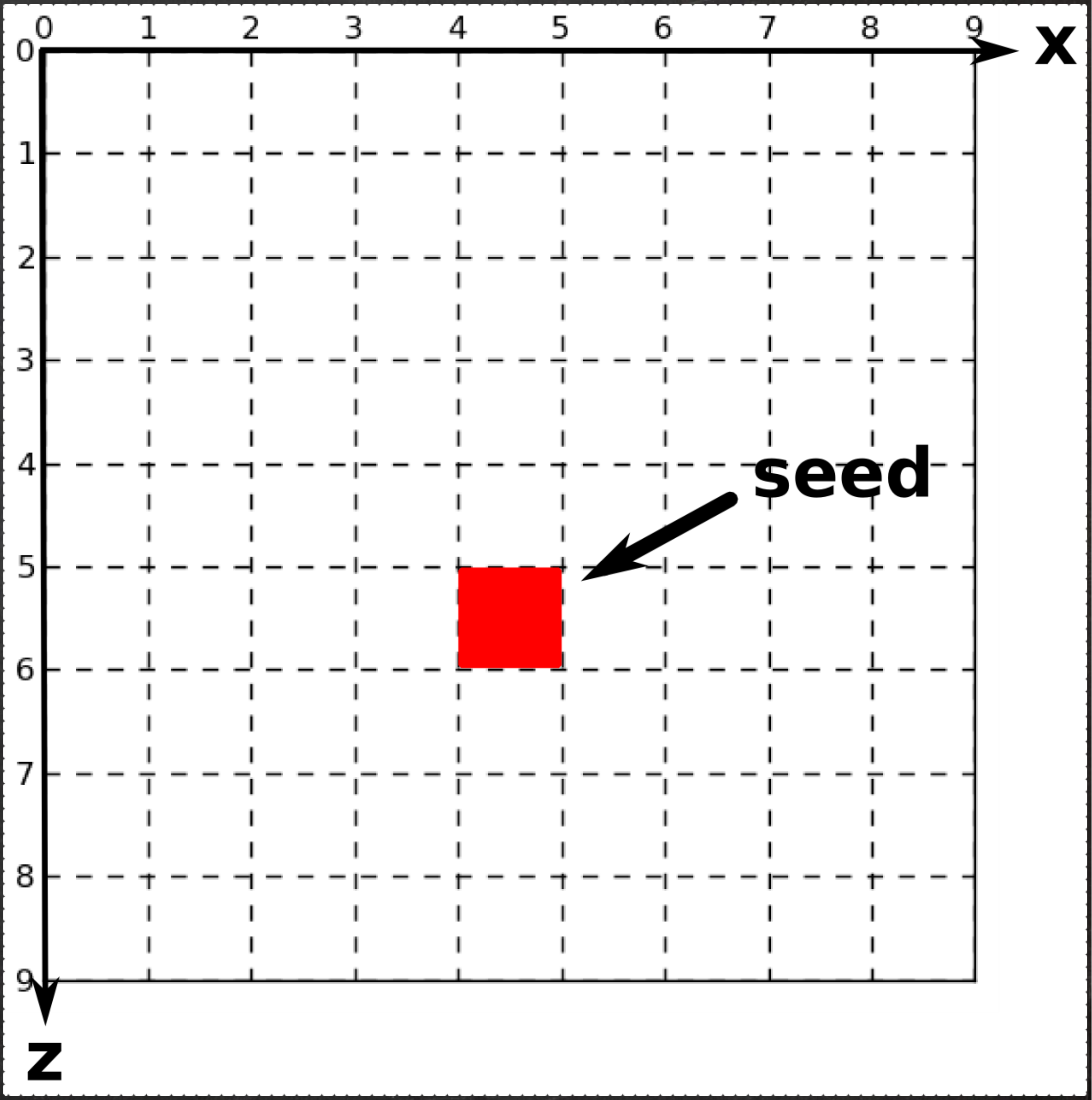
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



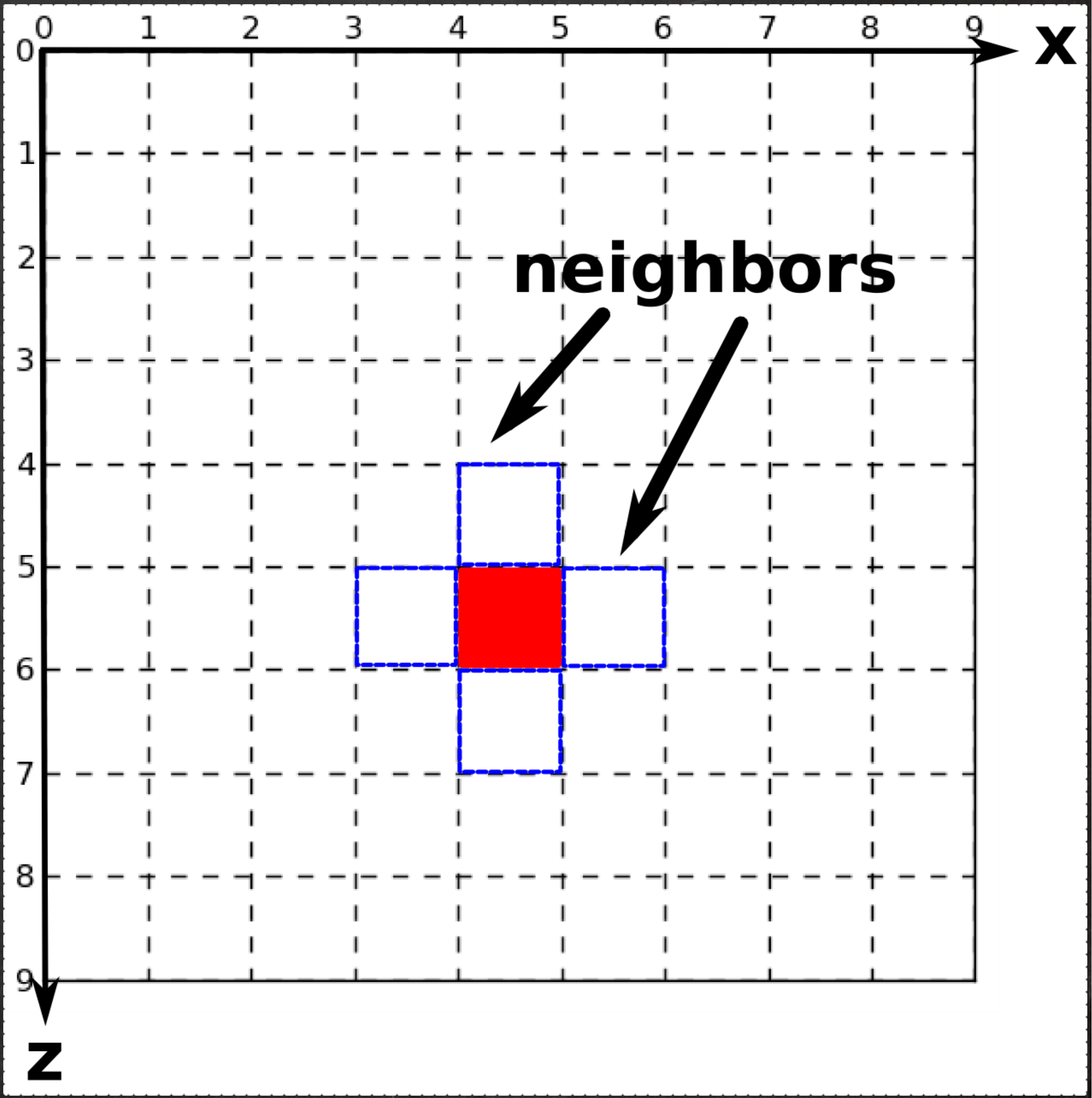
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



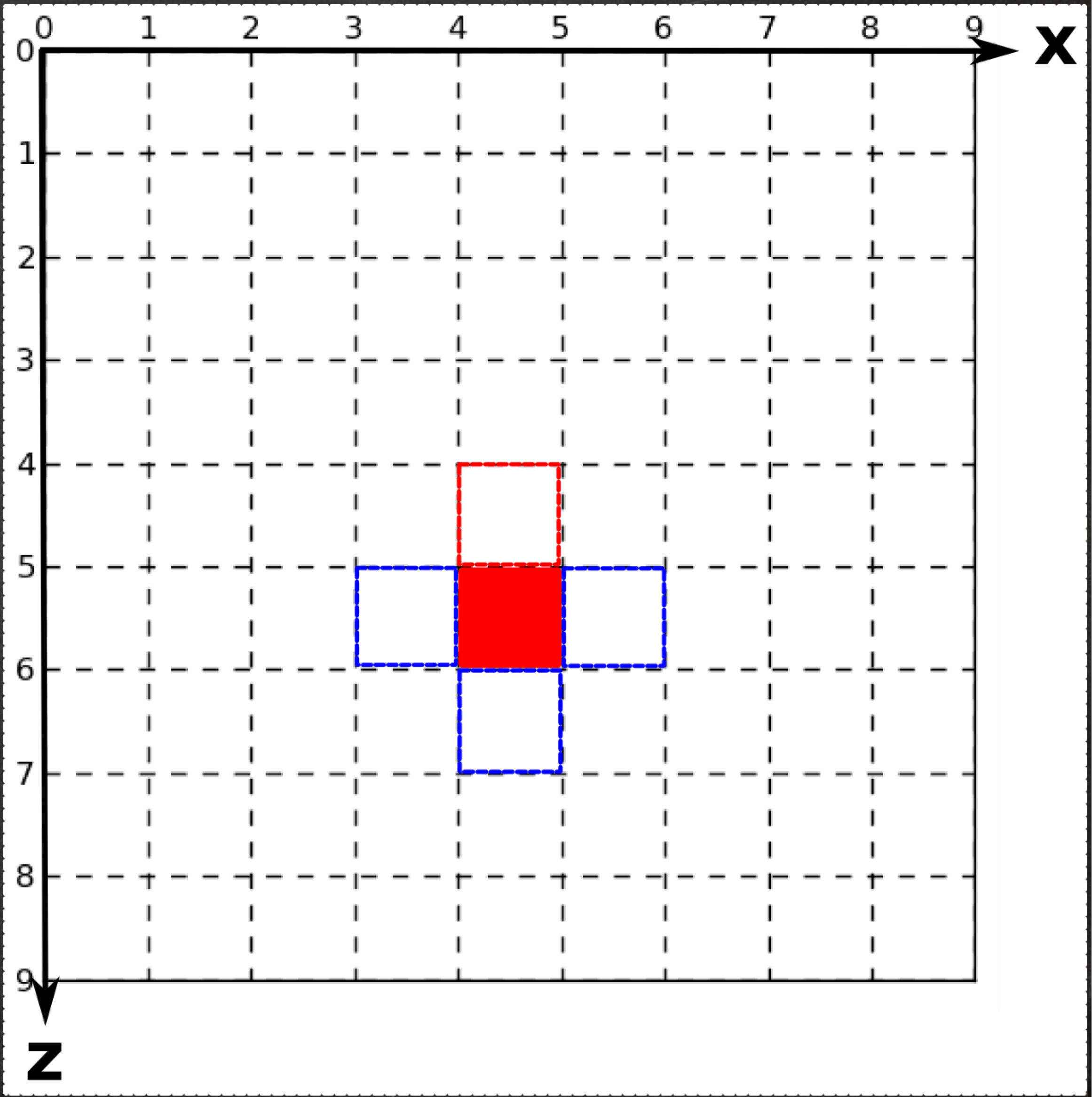
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



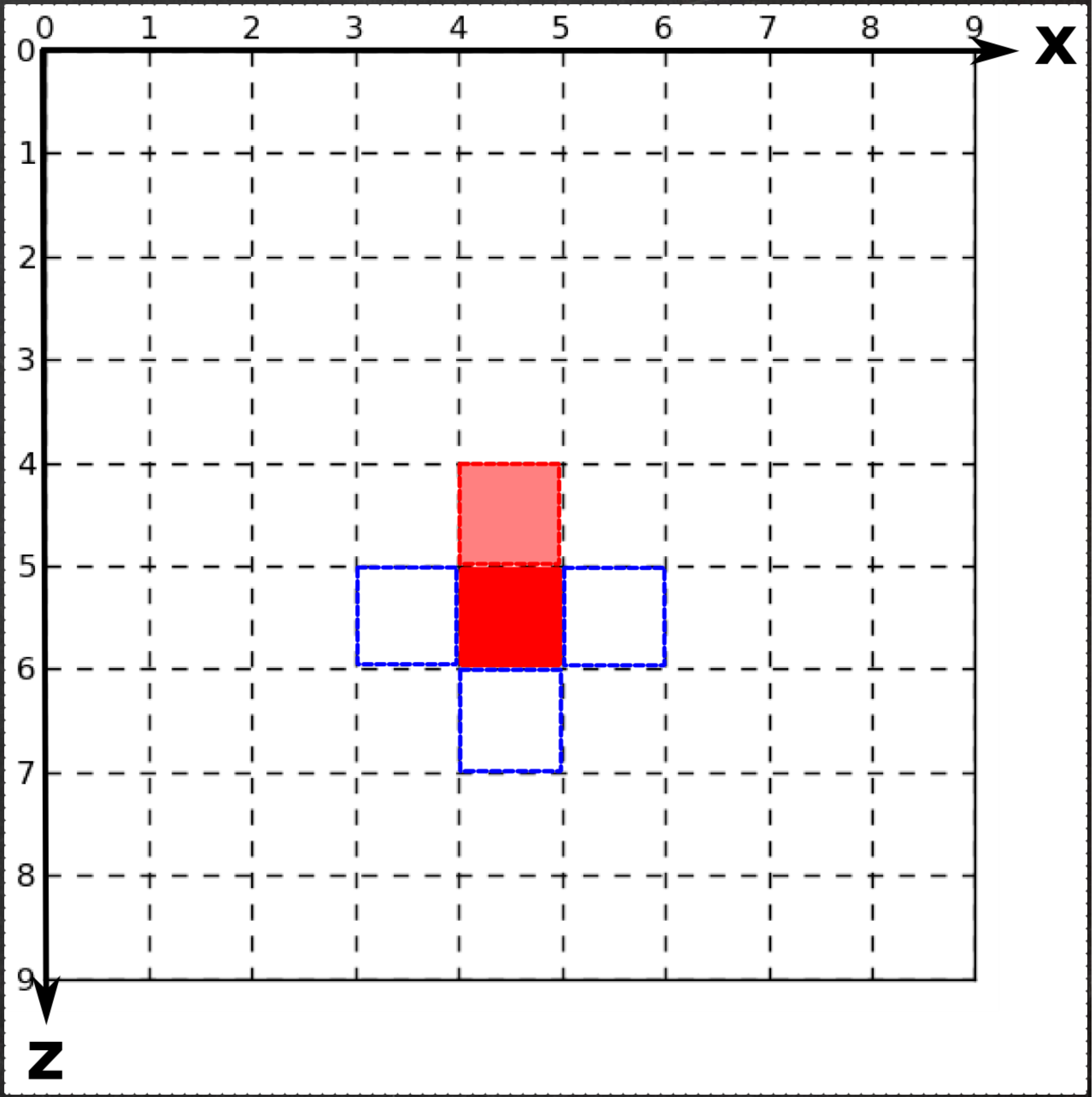
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
  for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
        best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



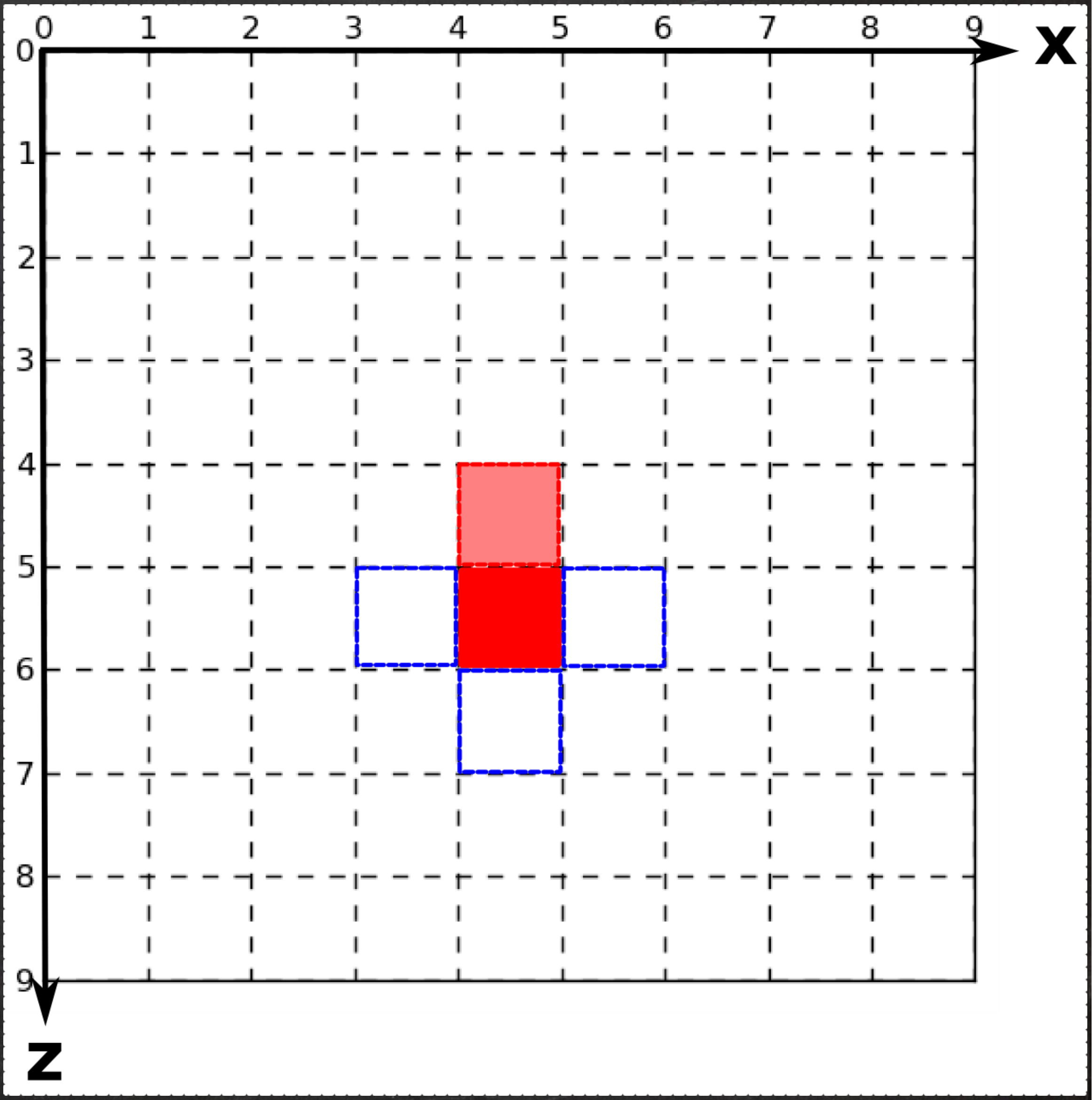
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



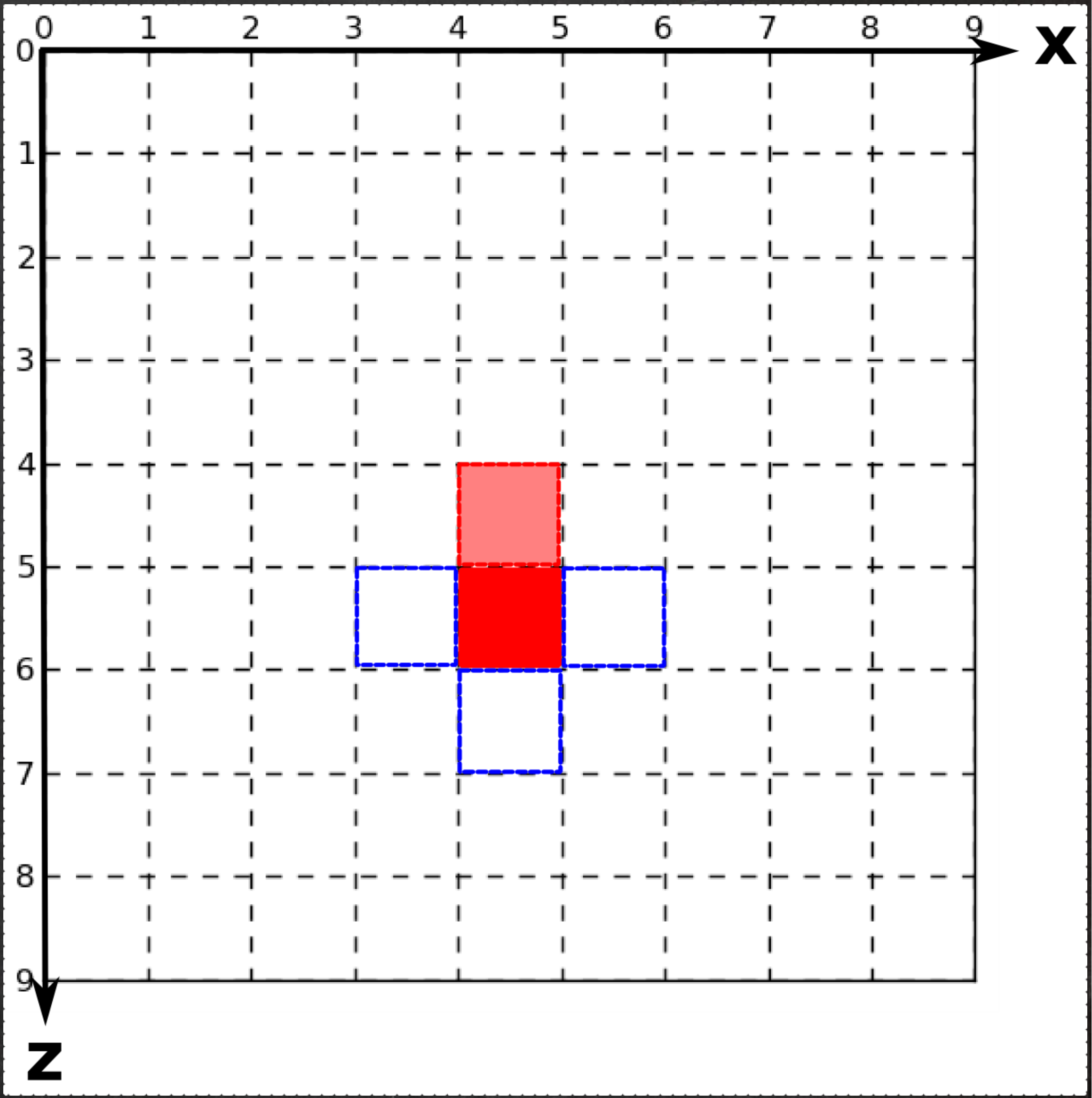
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



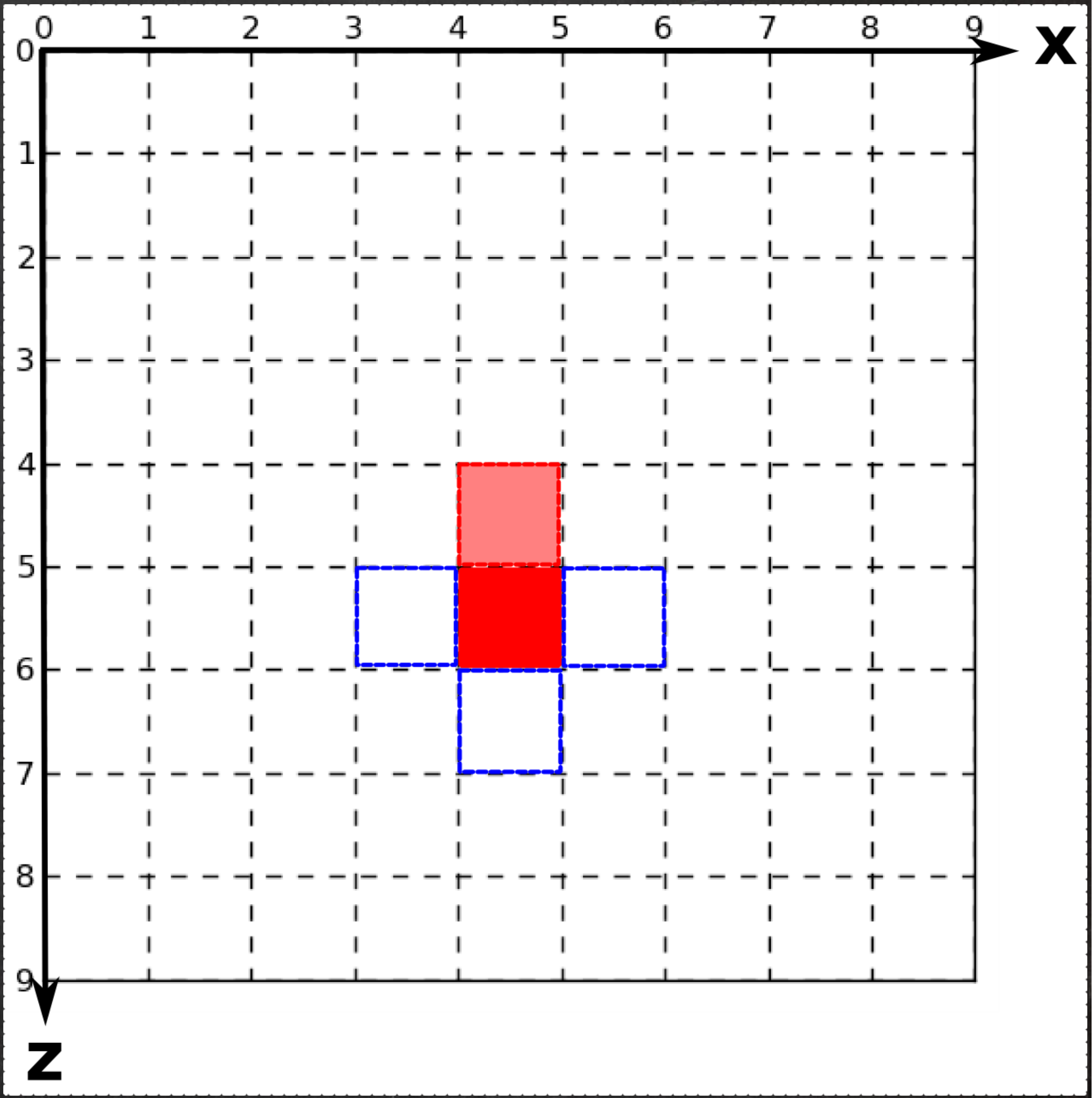
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



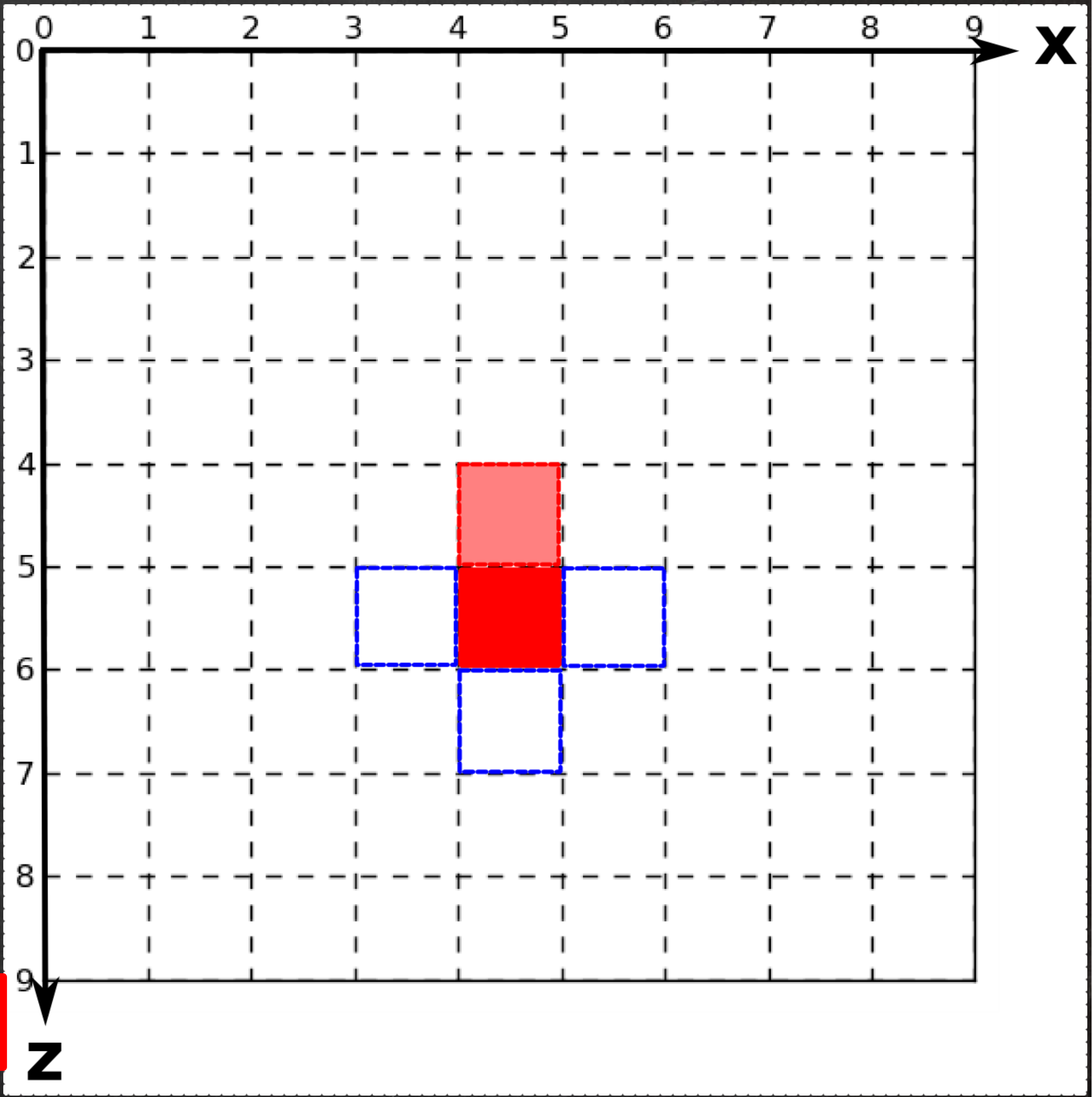
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



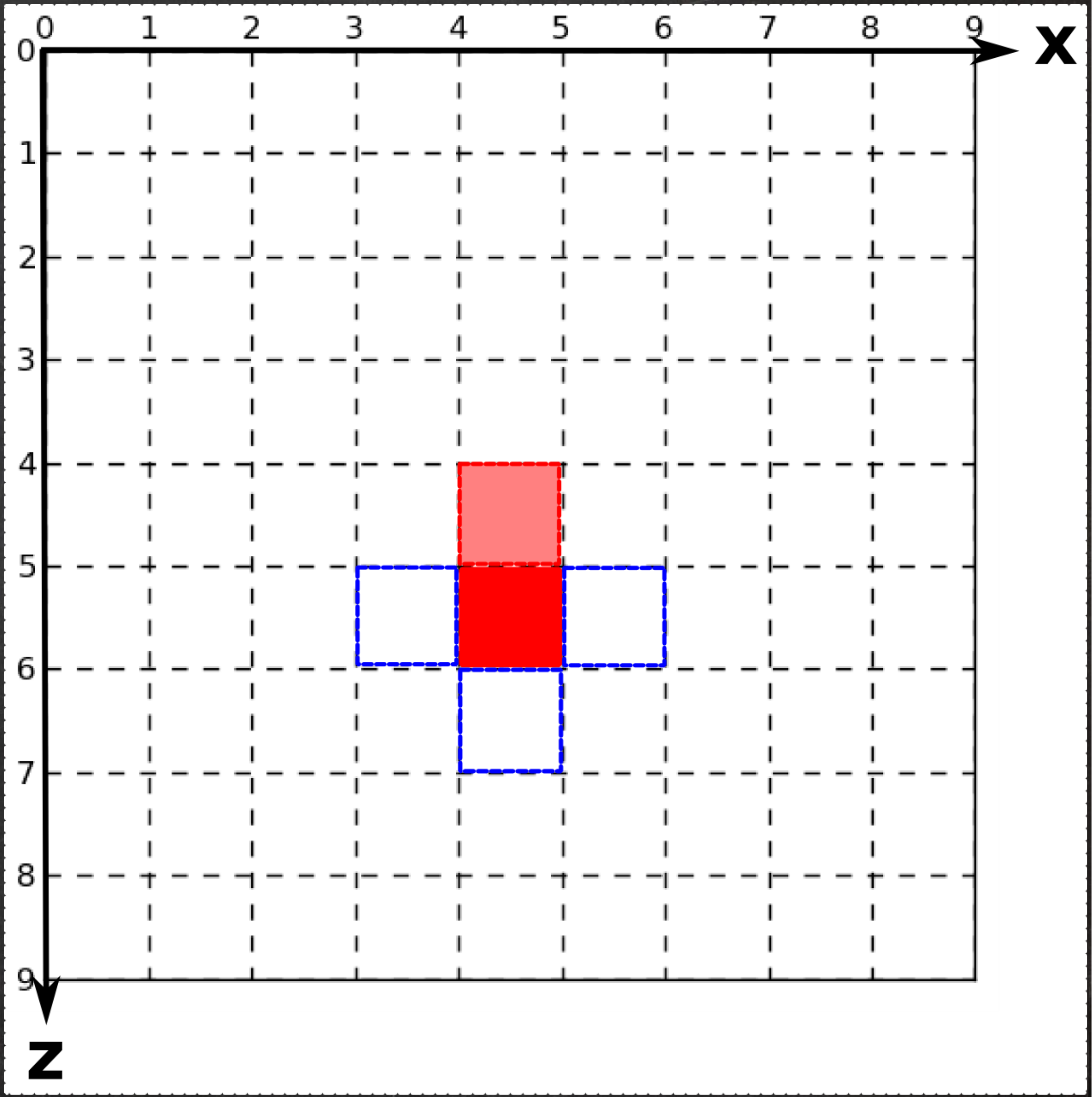
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



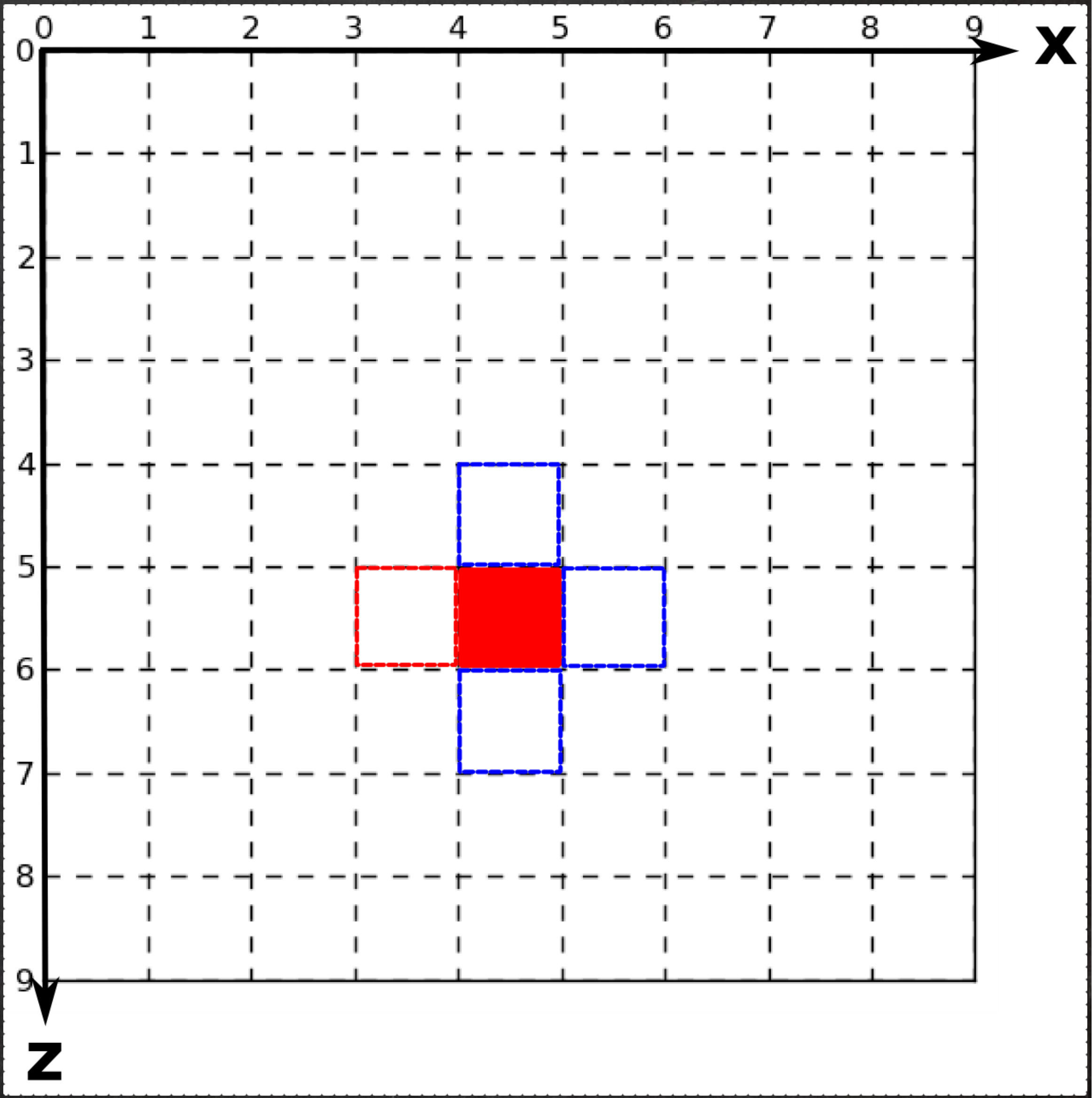
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



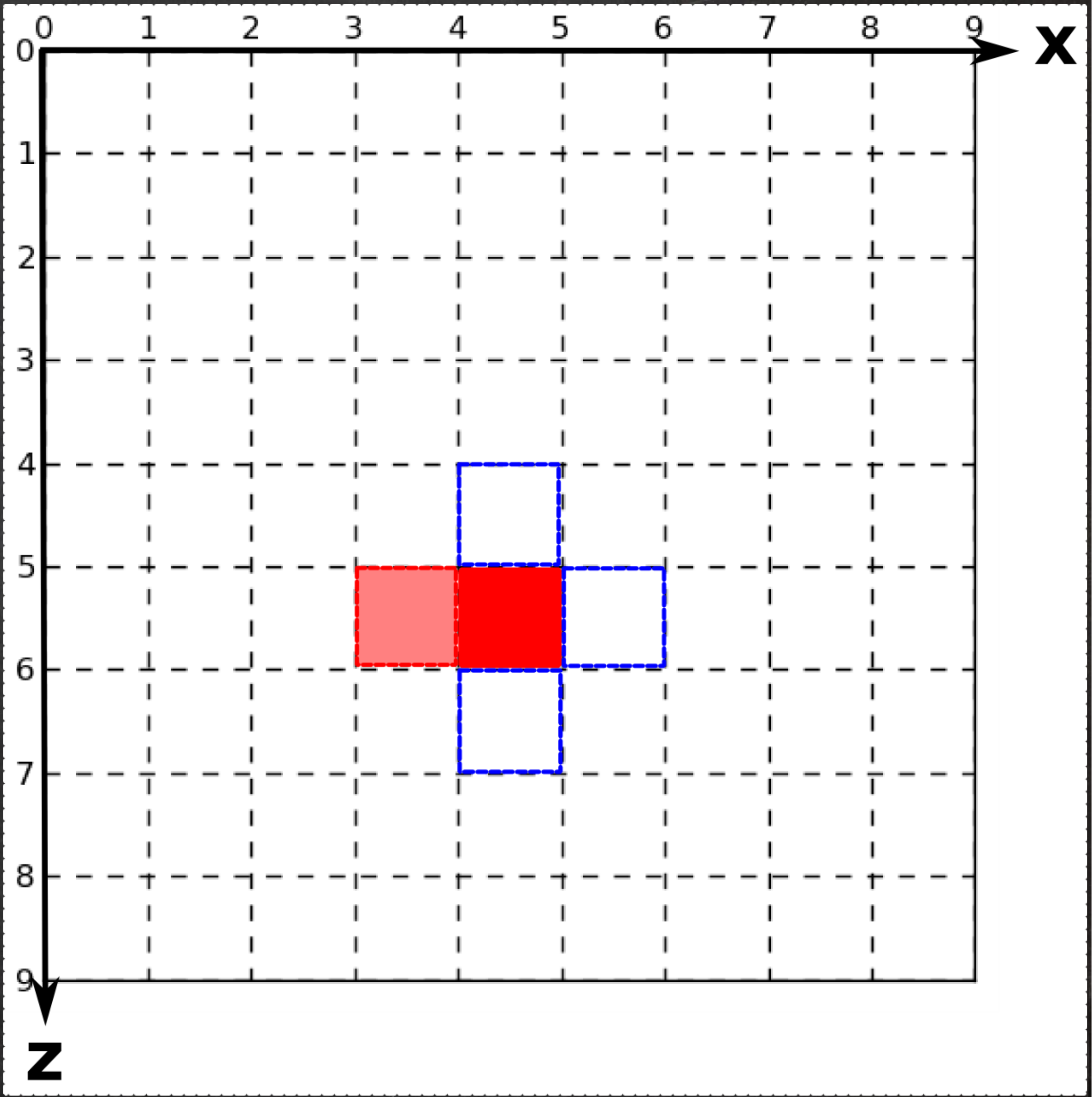
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



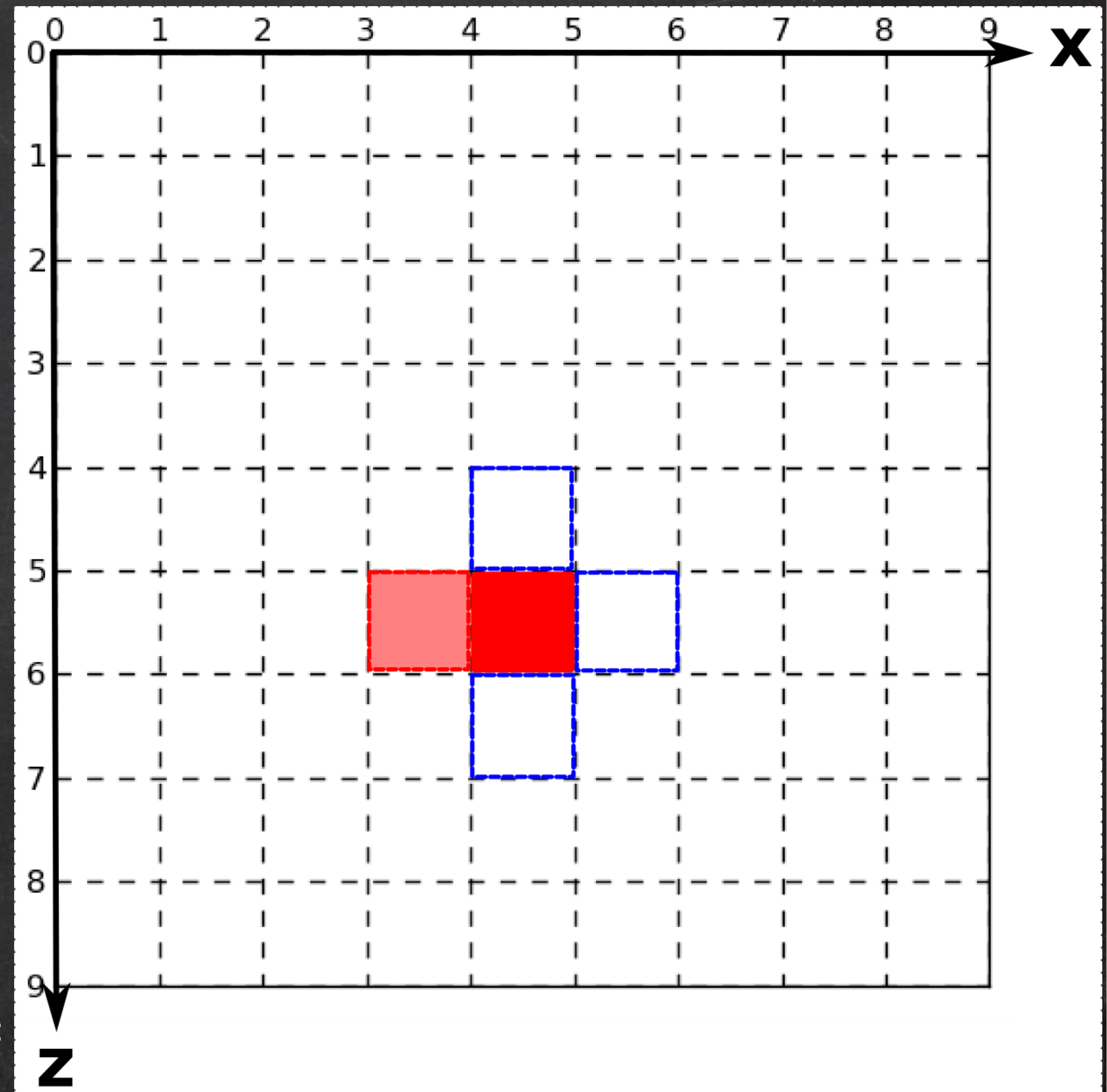
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



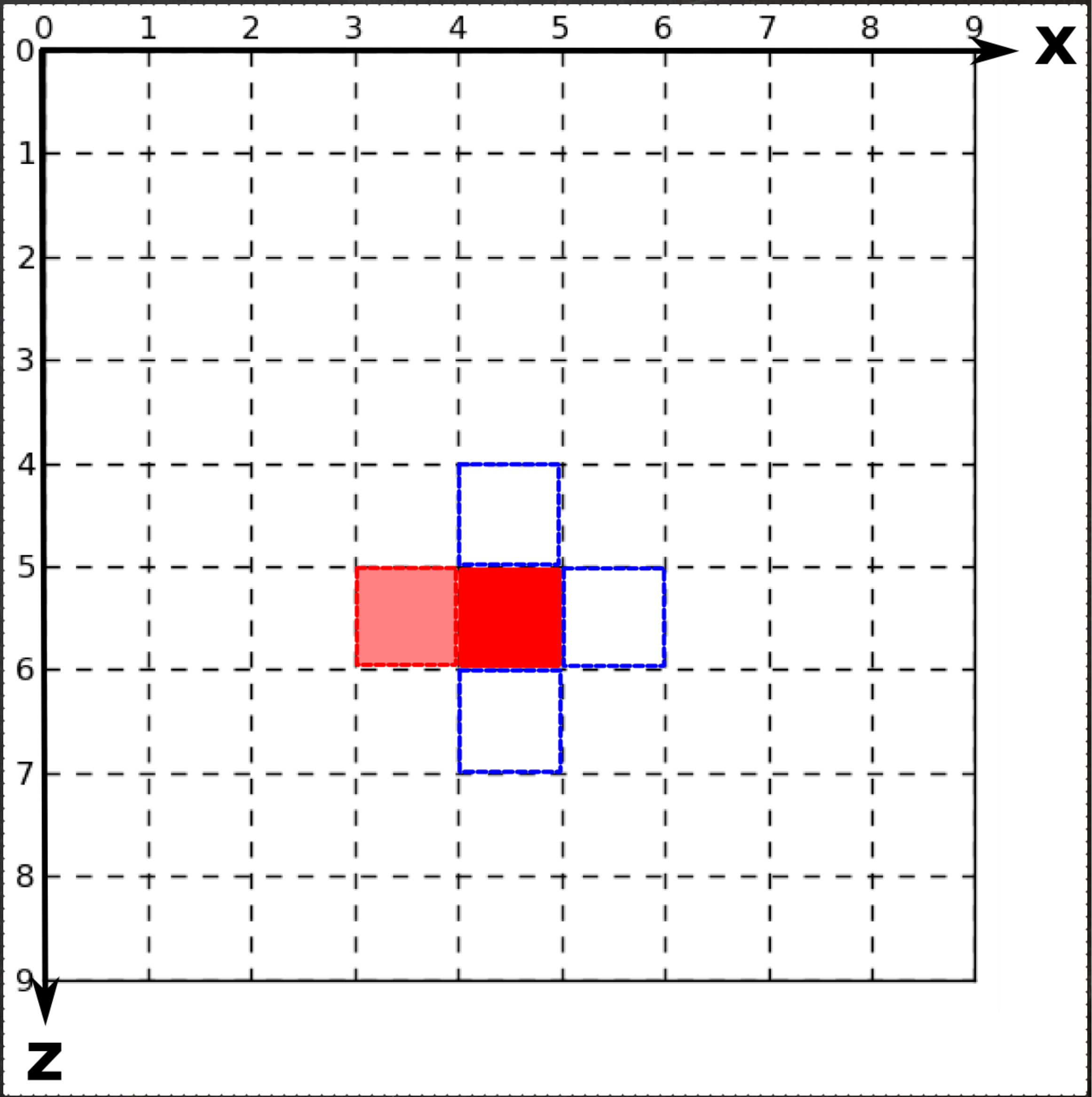
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



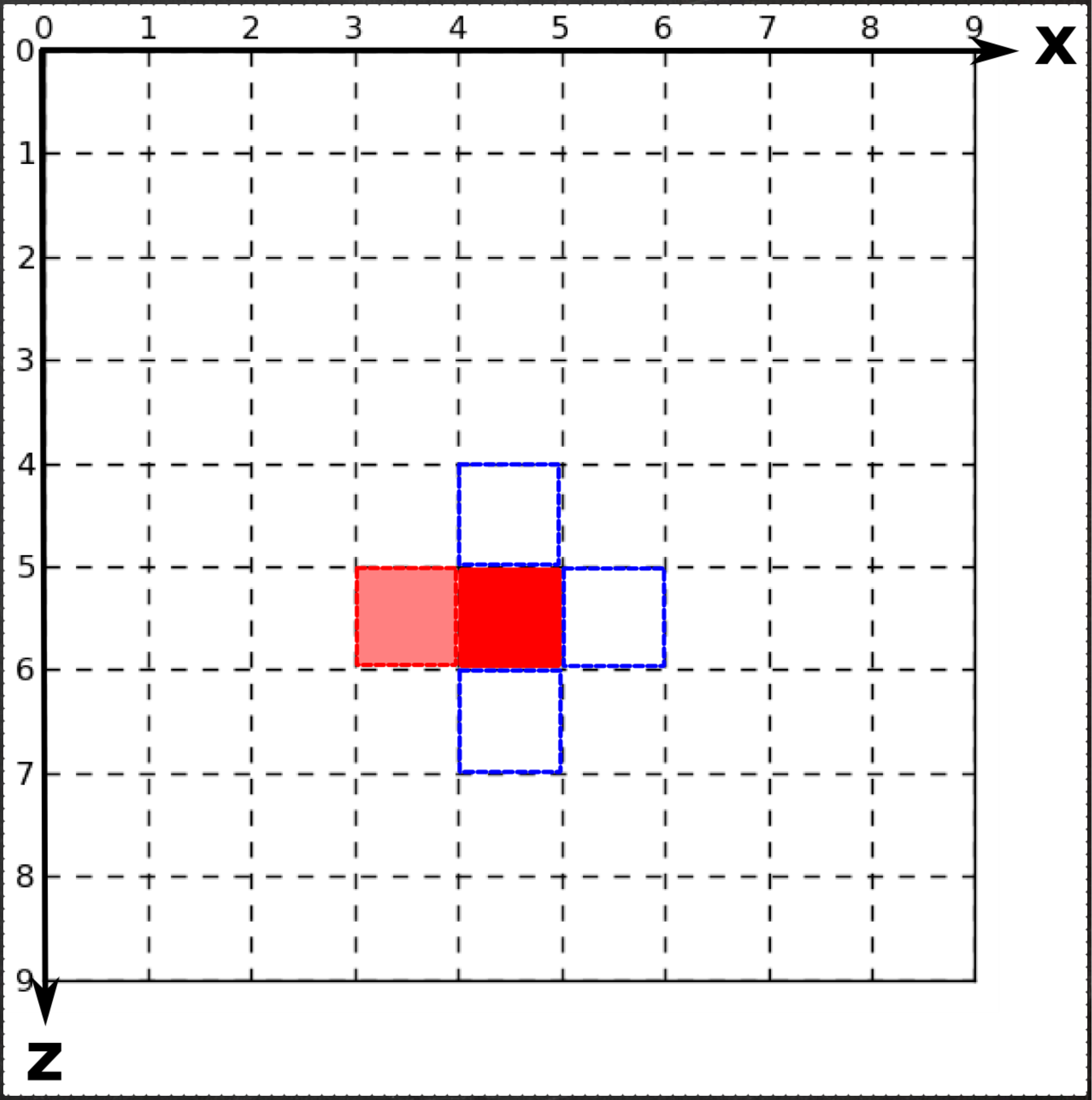
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



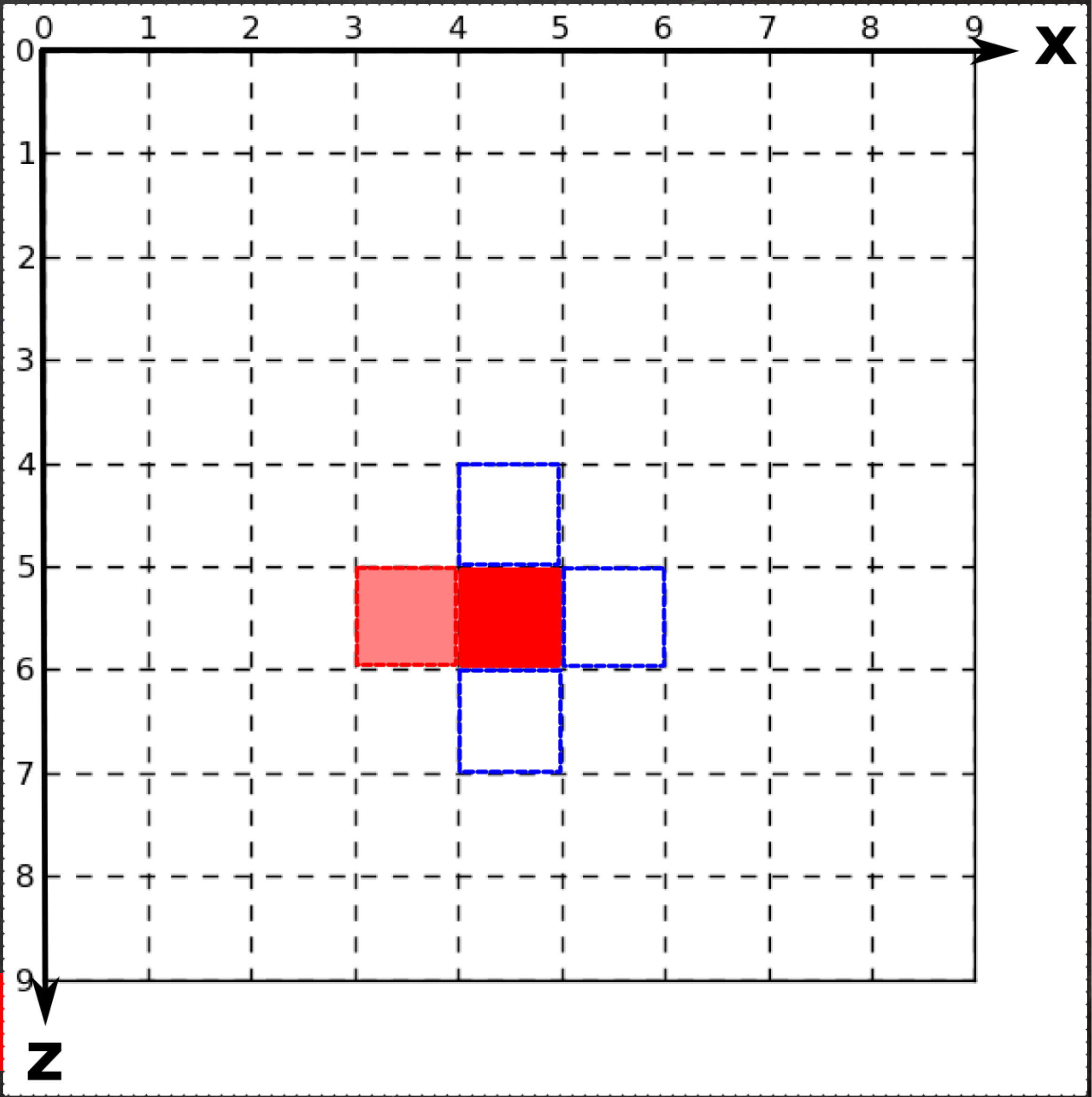
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



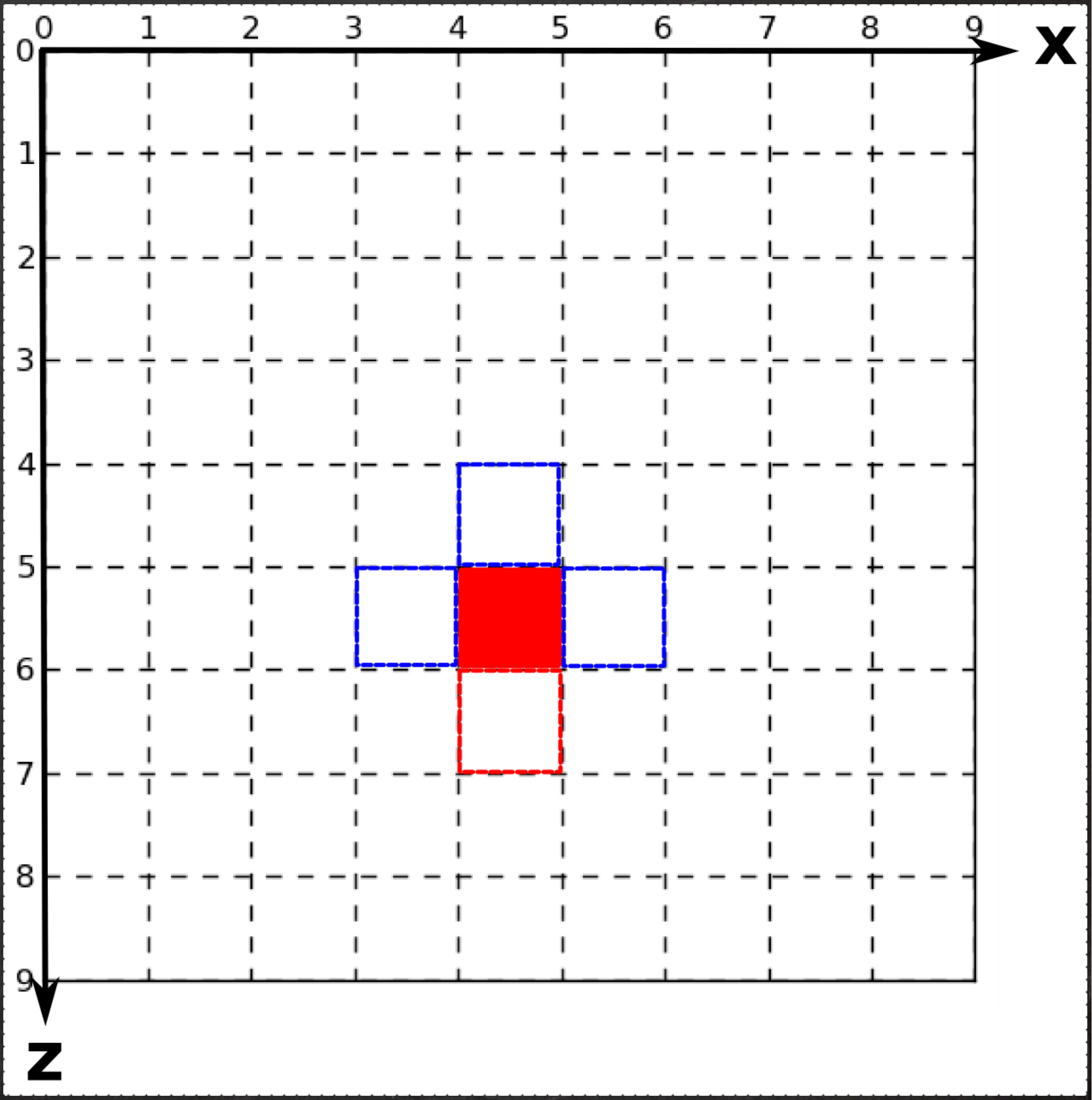
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



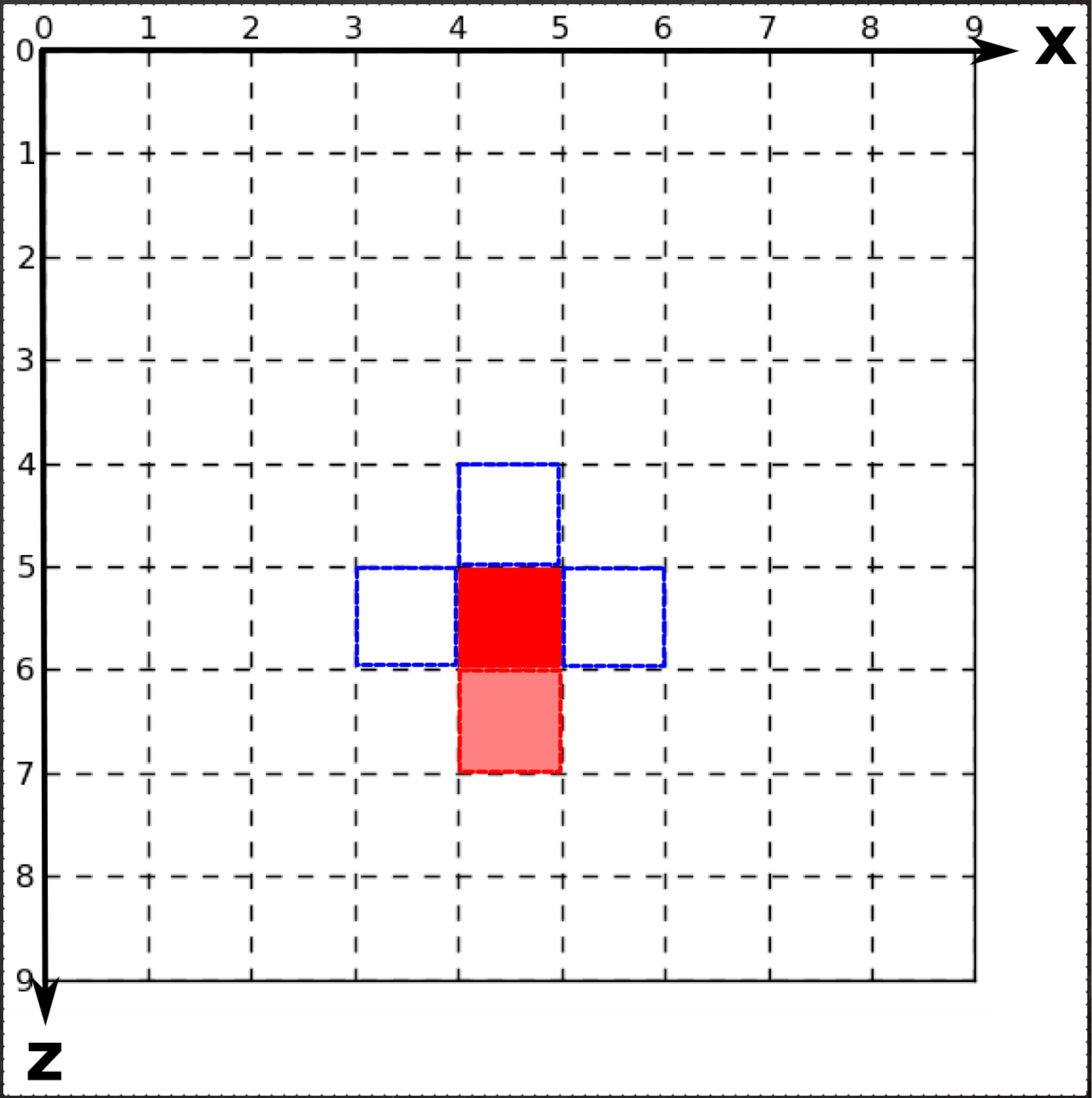
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



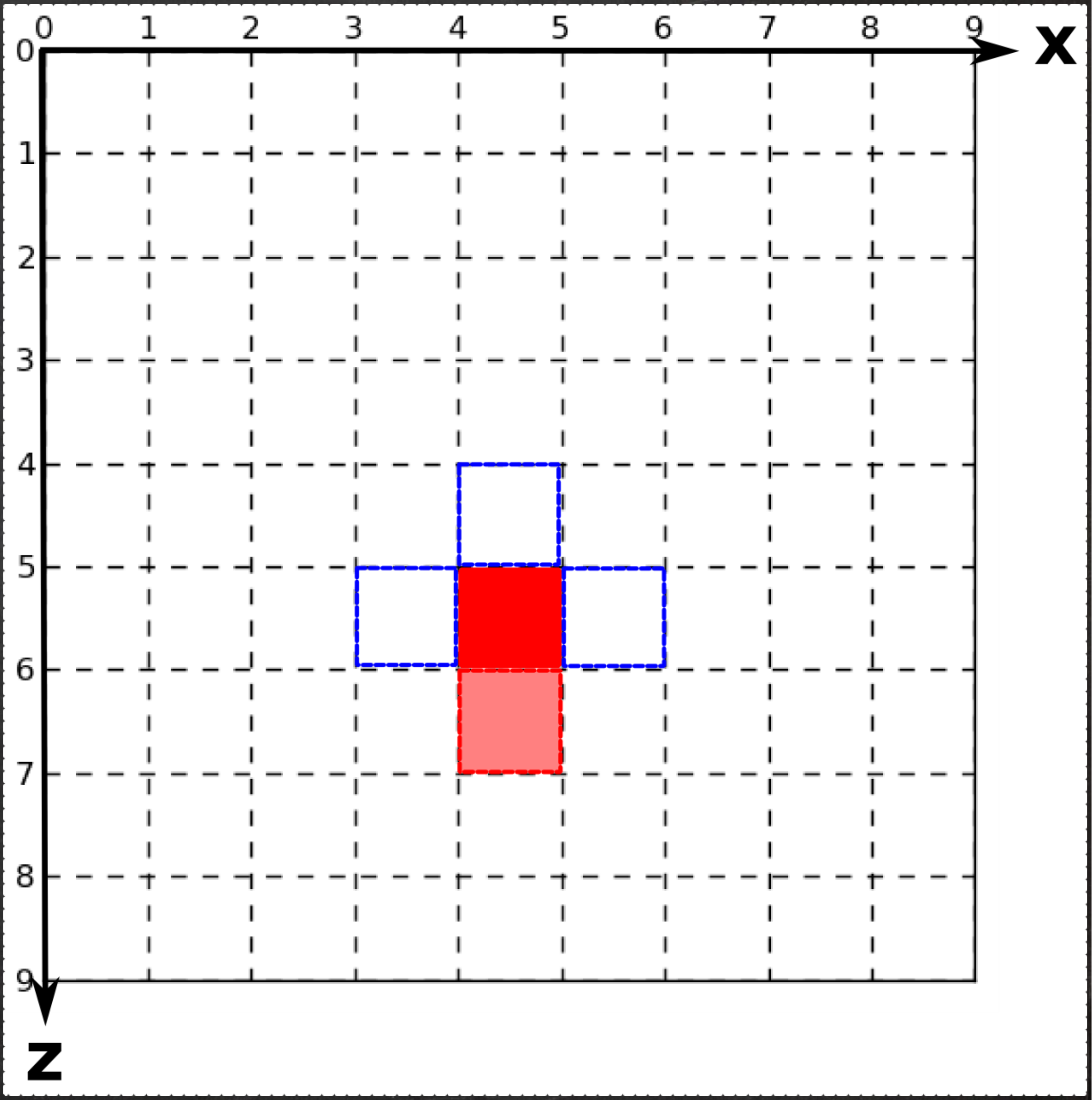
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



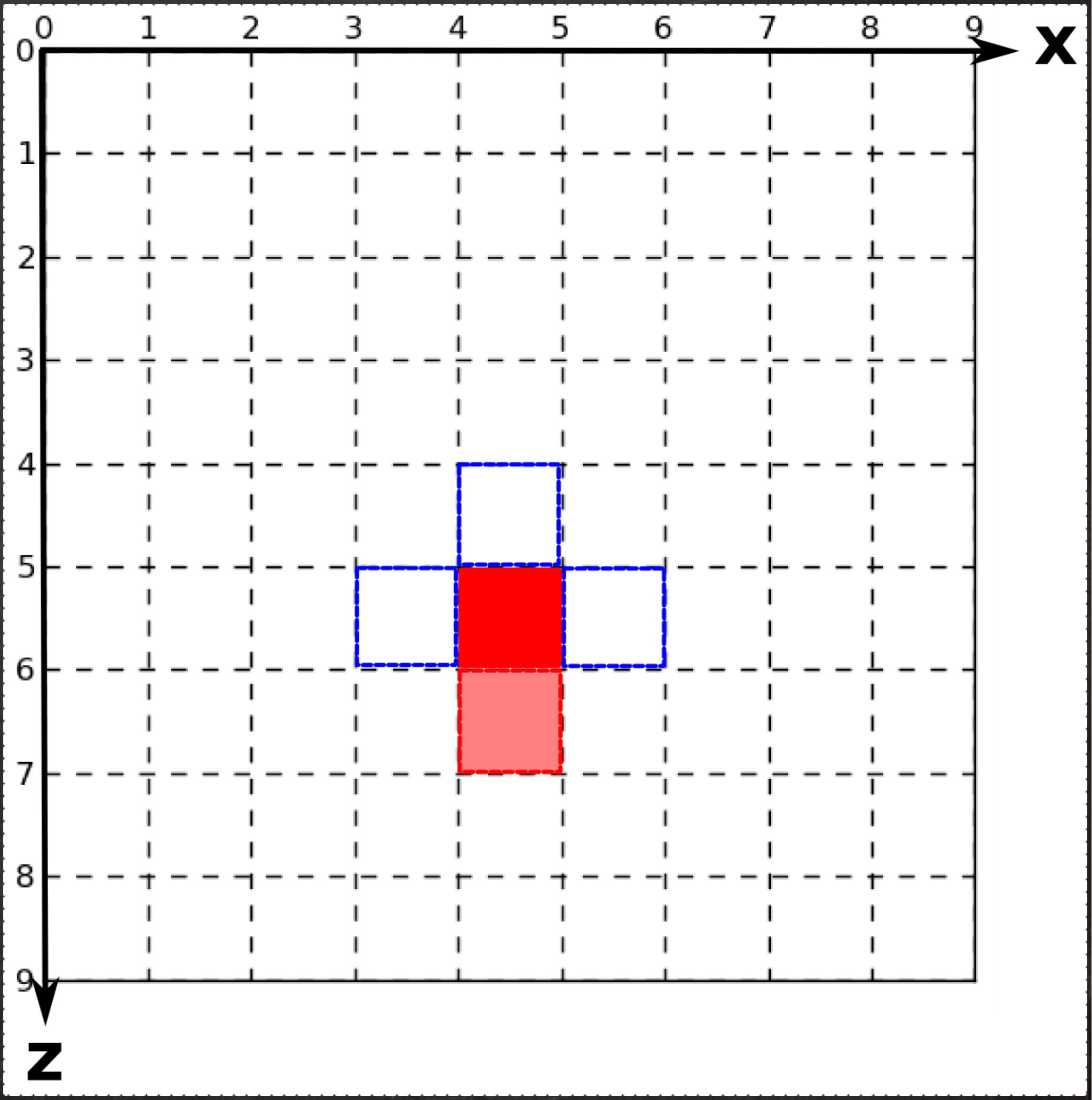
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



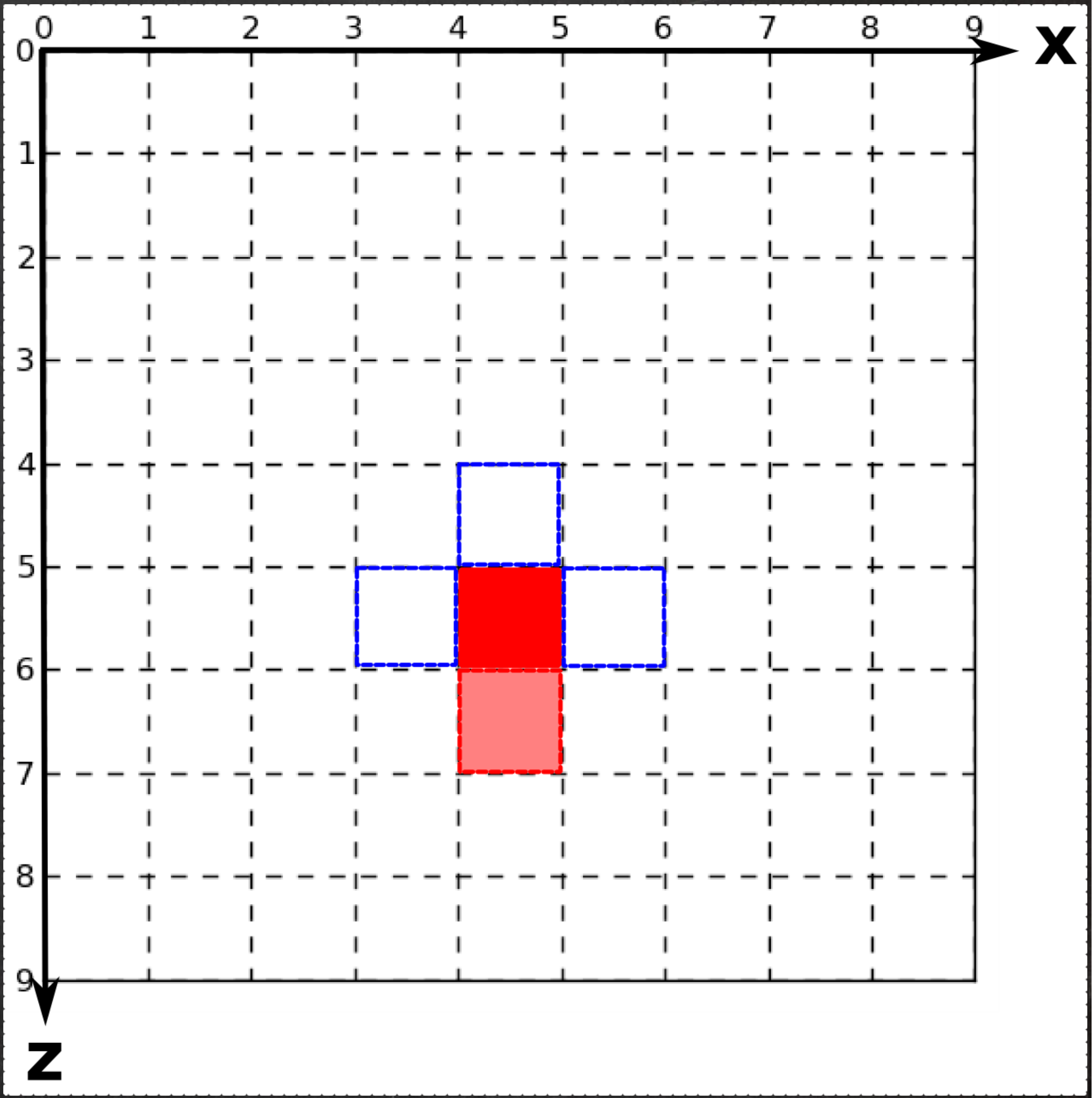
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



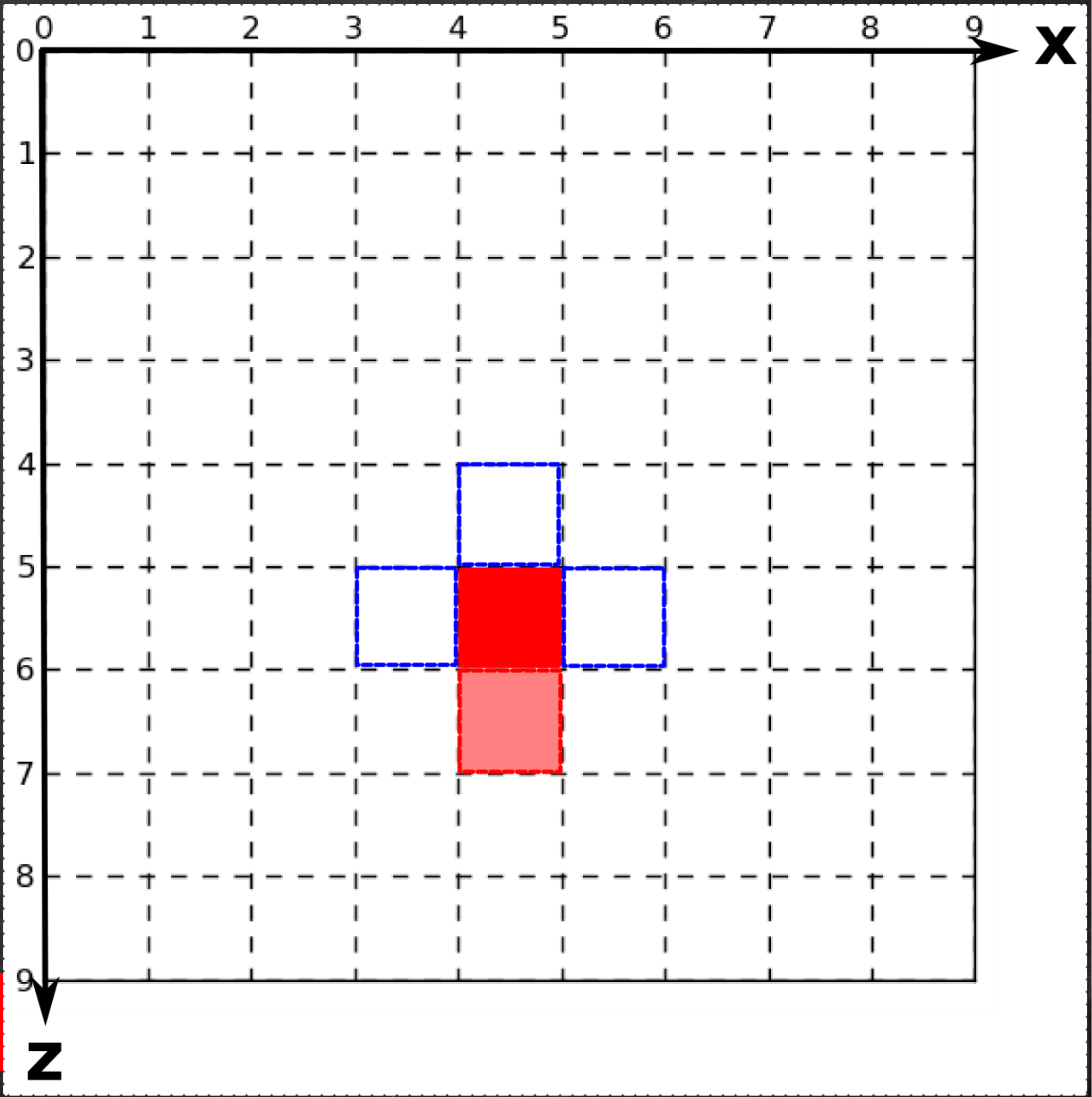
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



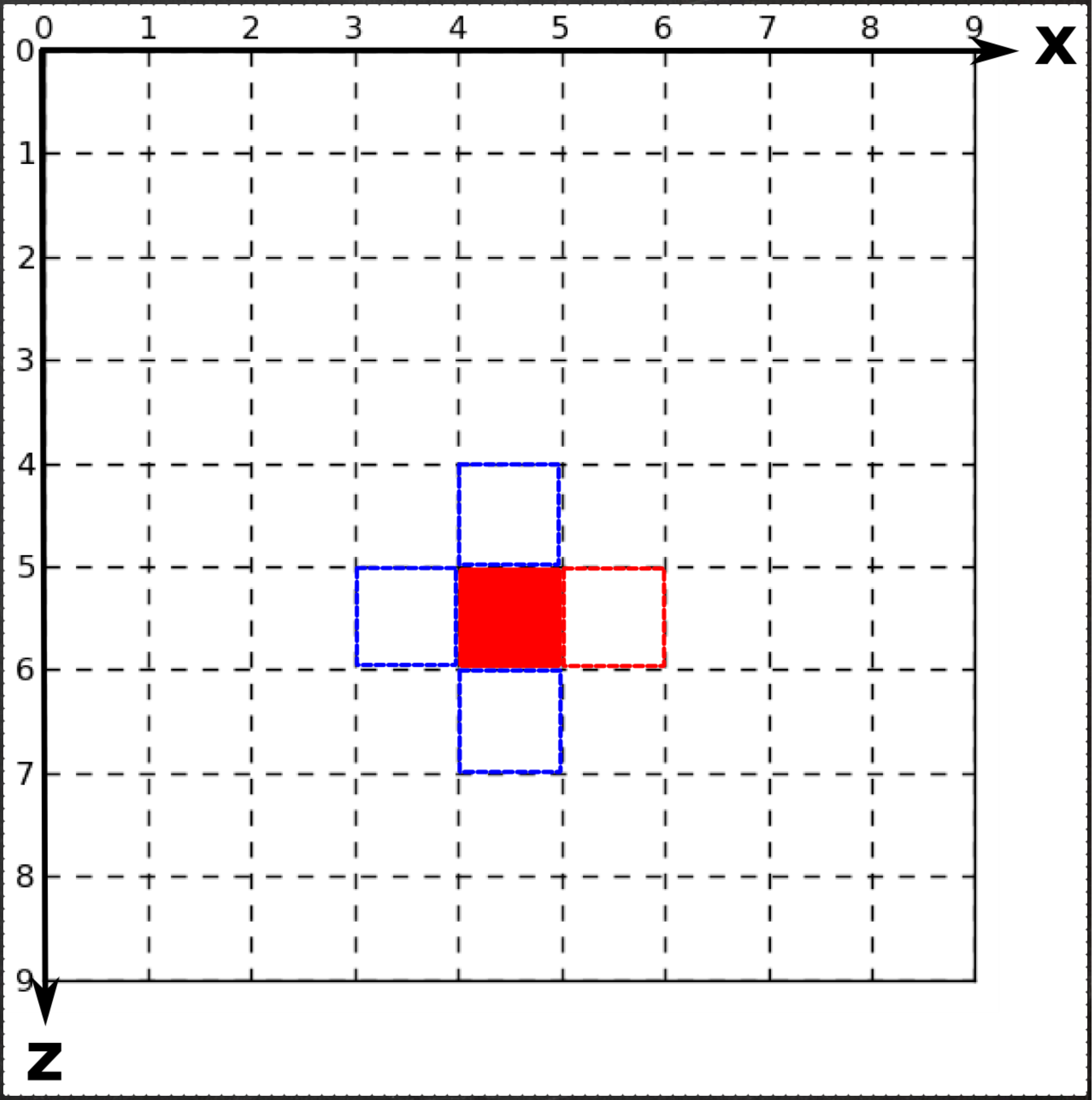
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



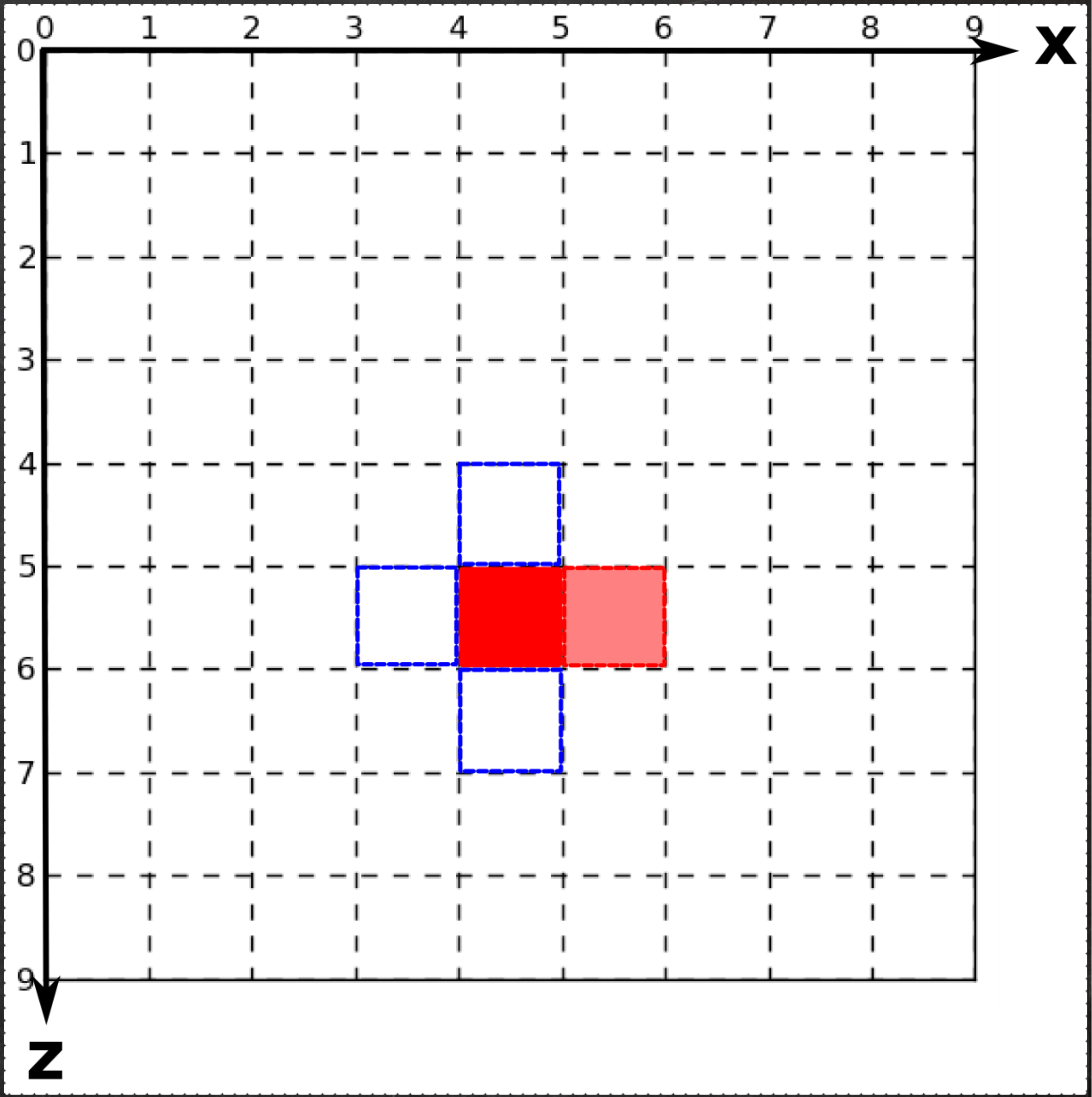
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



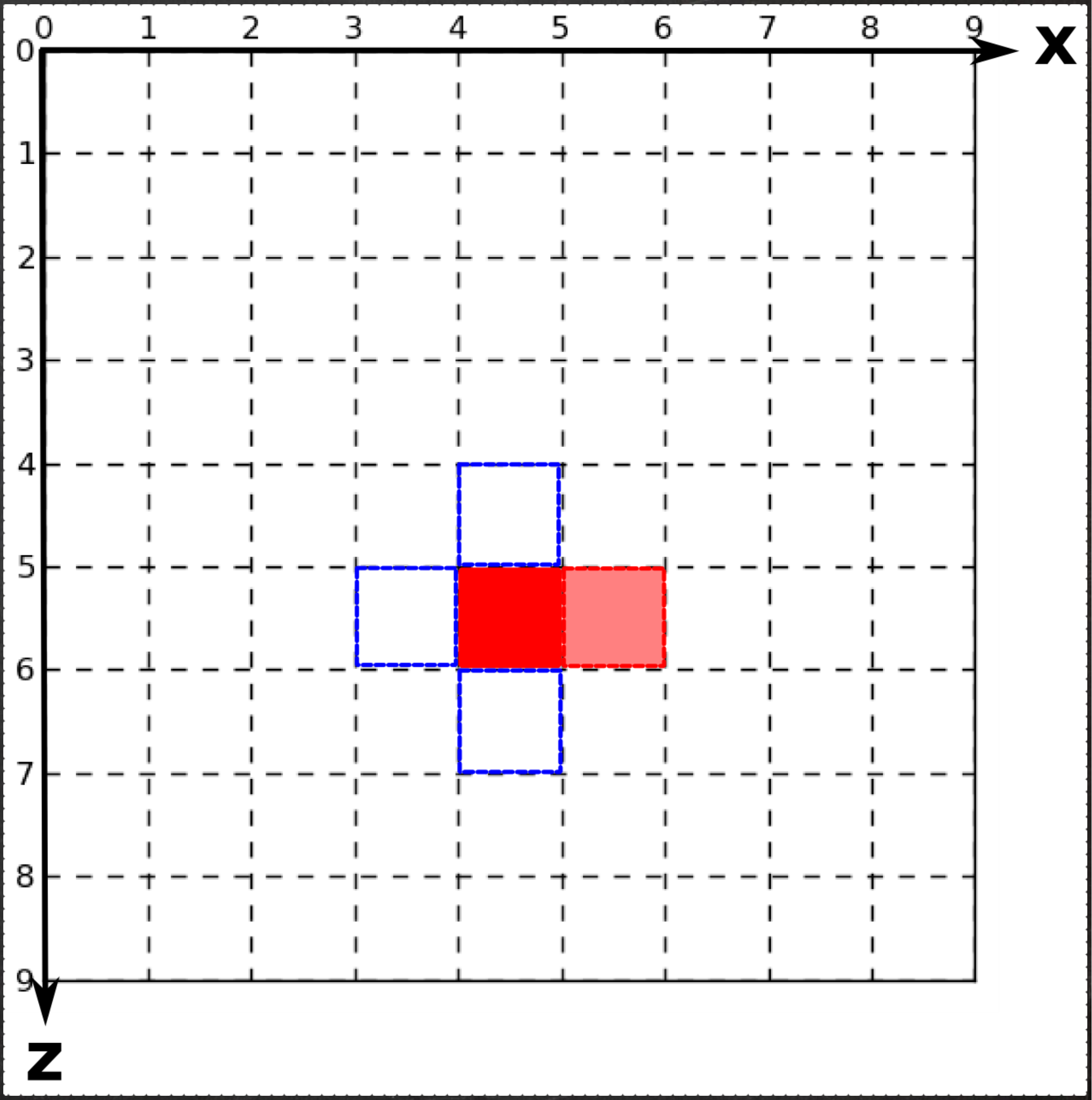
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



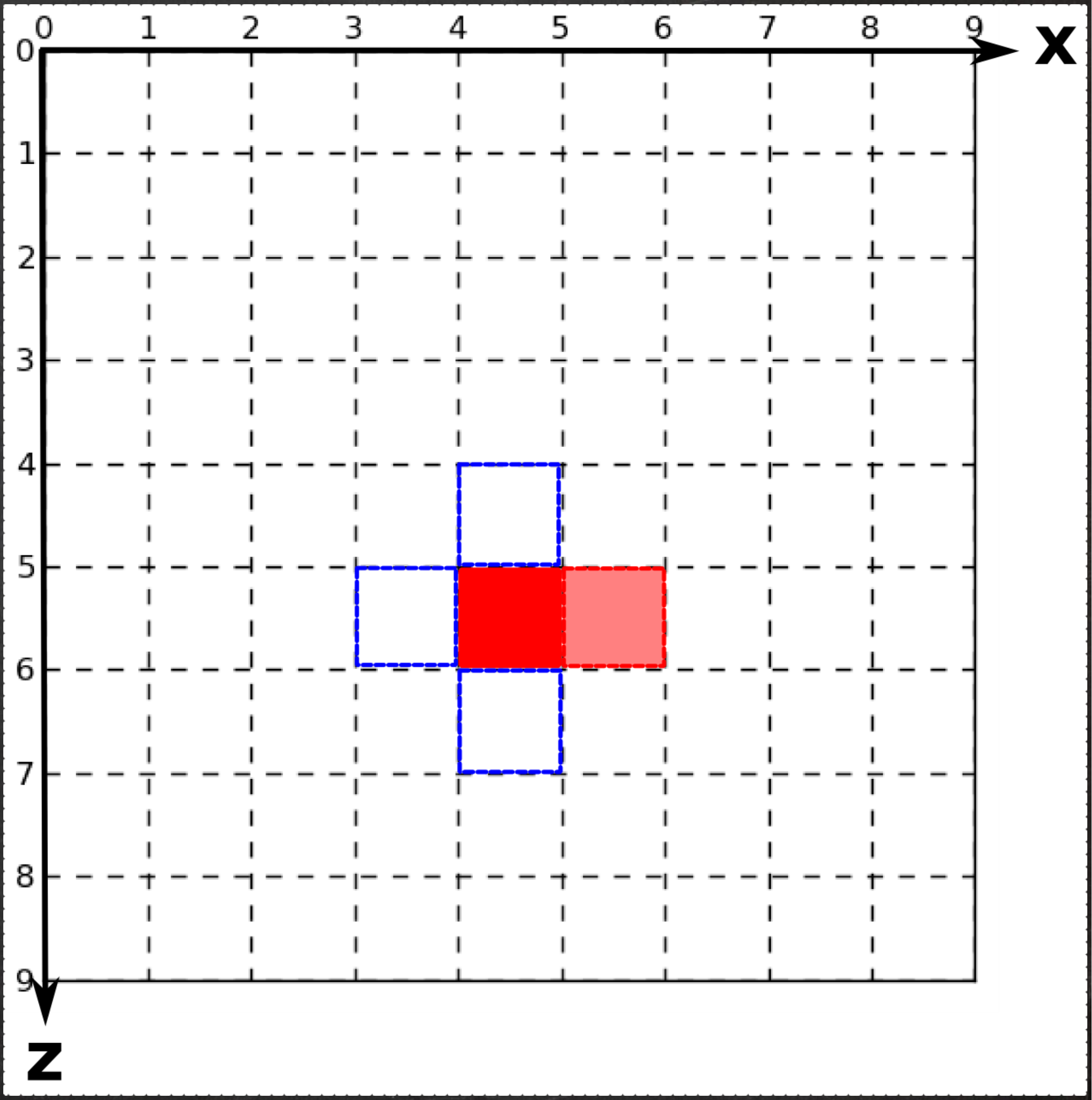
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



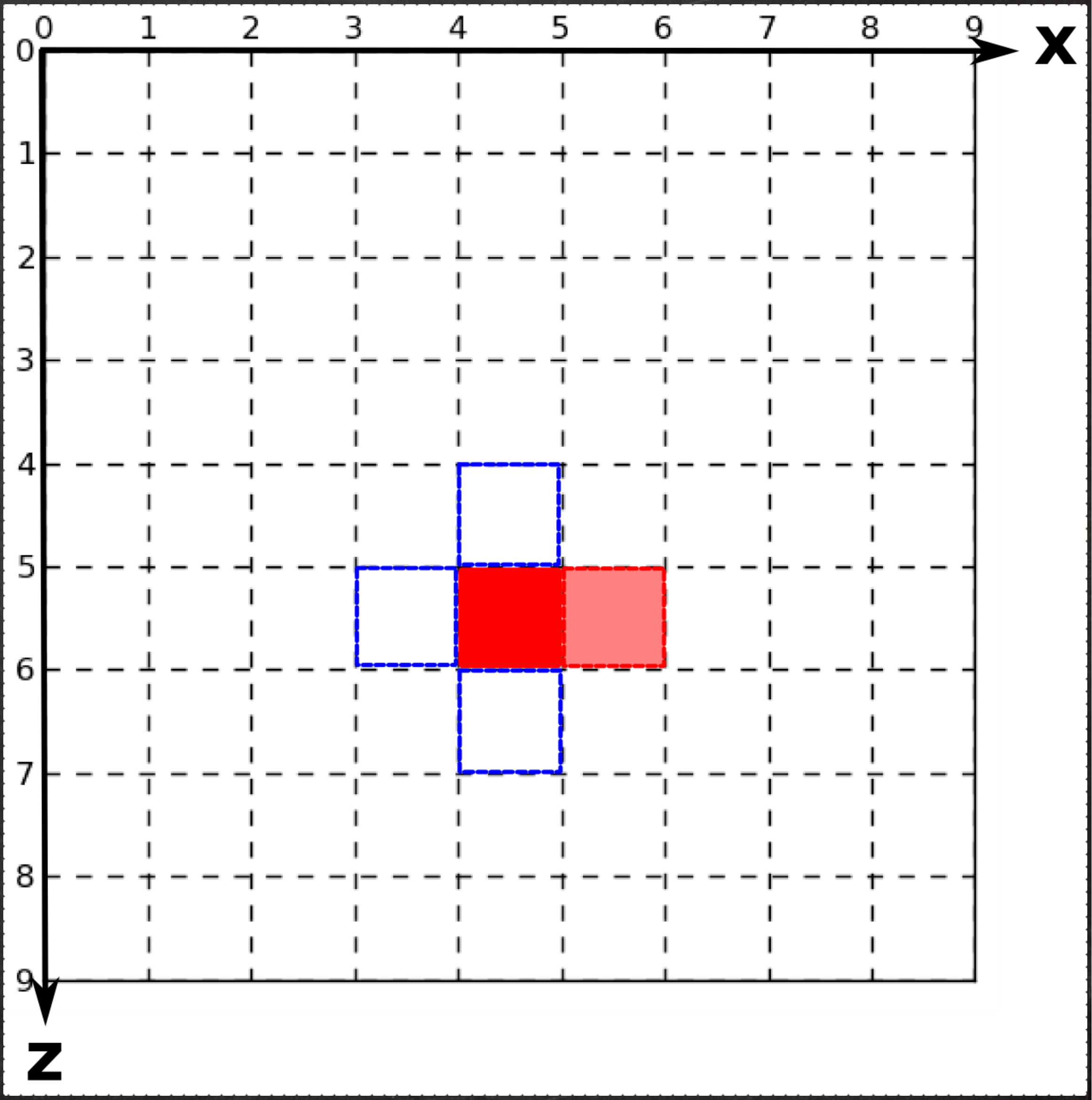
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



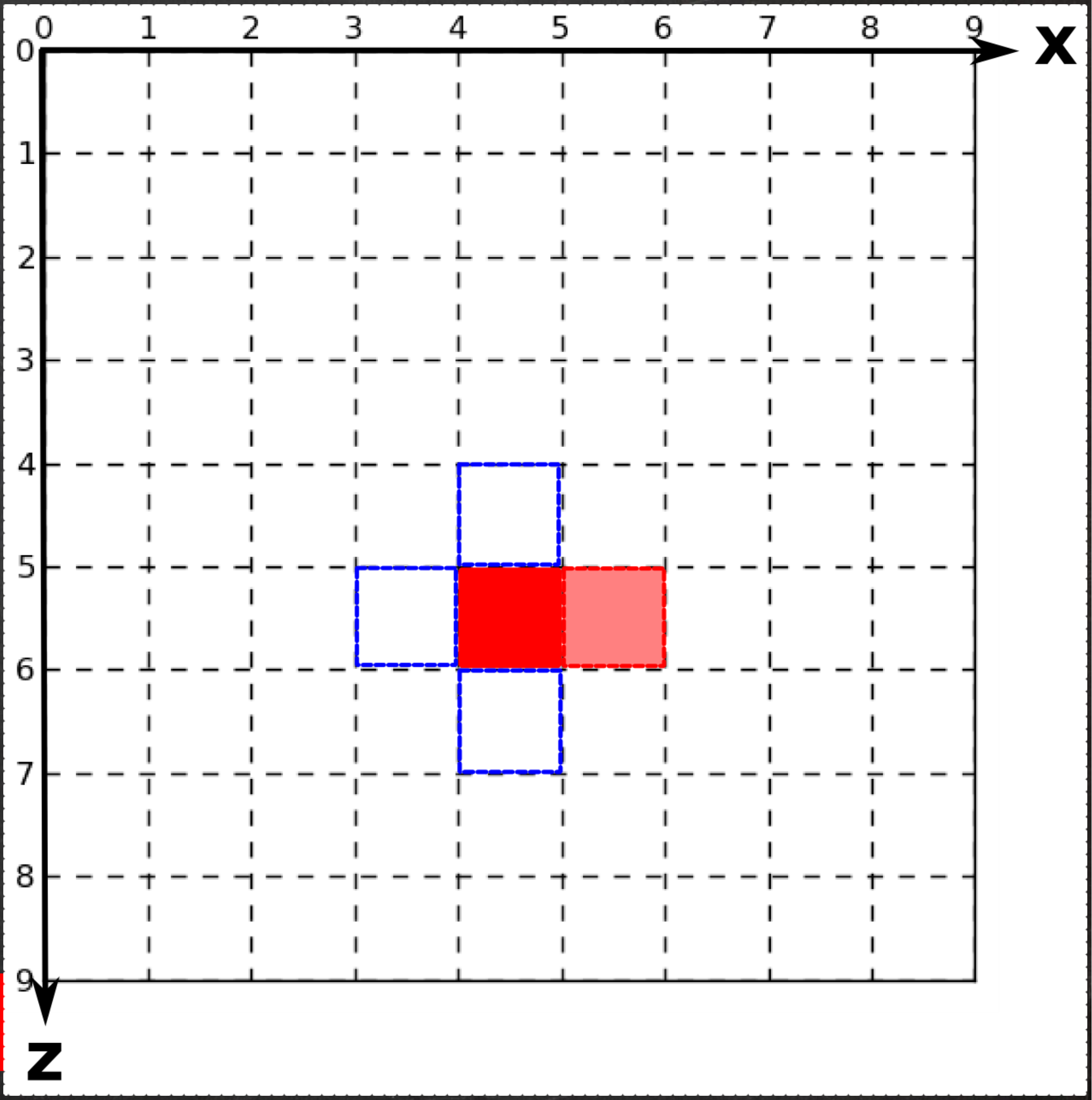
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
    for n in neighbors[s]:
      p[n] = dens[s]
      newr -= p[n]*Gtrans(n)
      misfit = norm(newr)
      reg = regularizer(p)
      if misfit + reg < best:
       best_n = n
    update_p_r(best_n)
    append_neighbors(best_n)
```



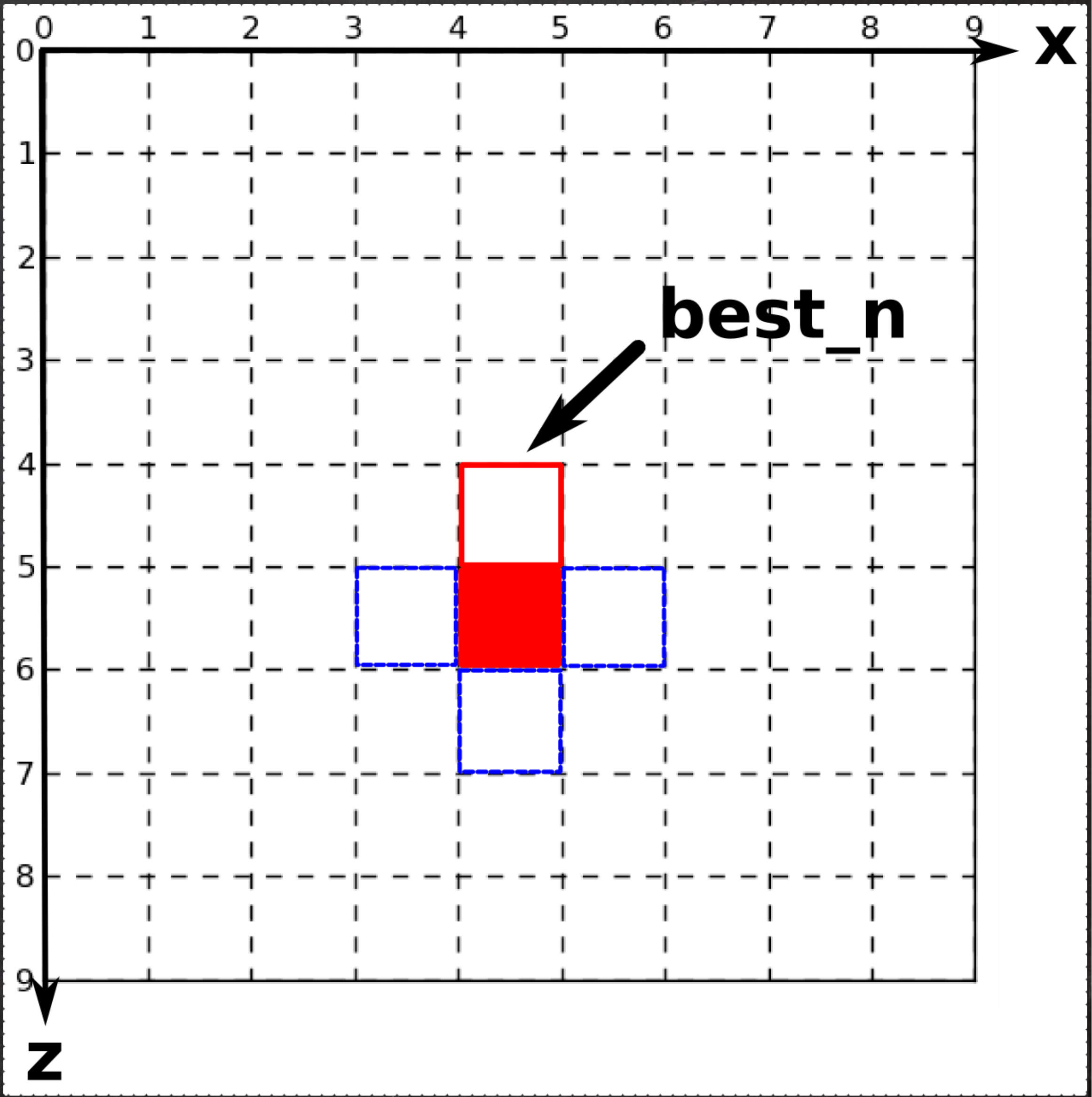
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



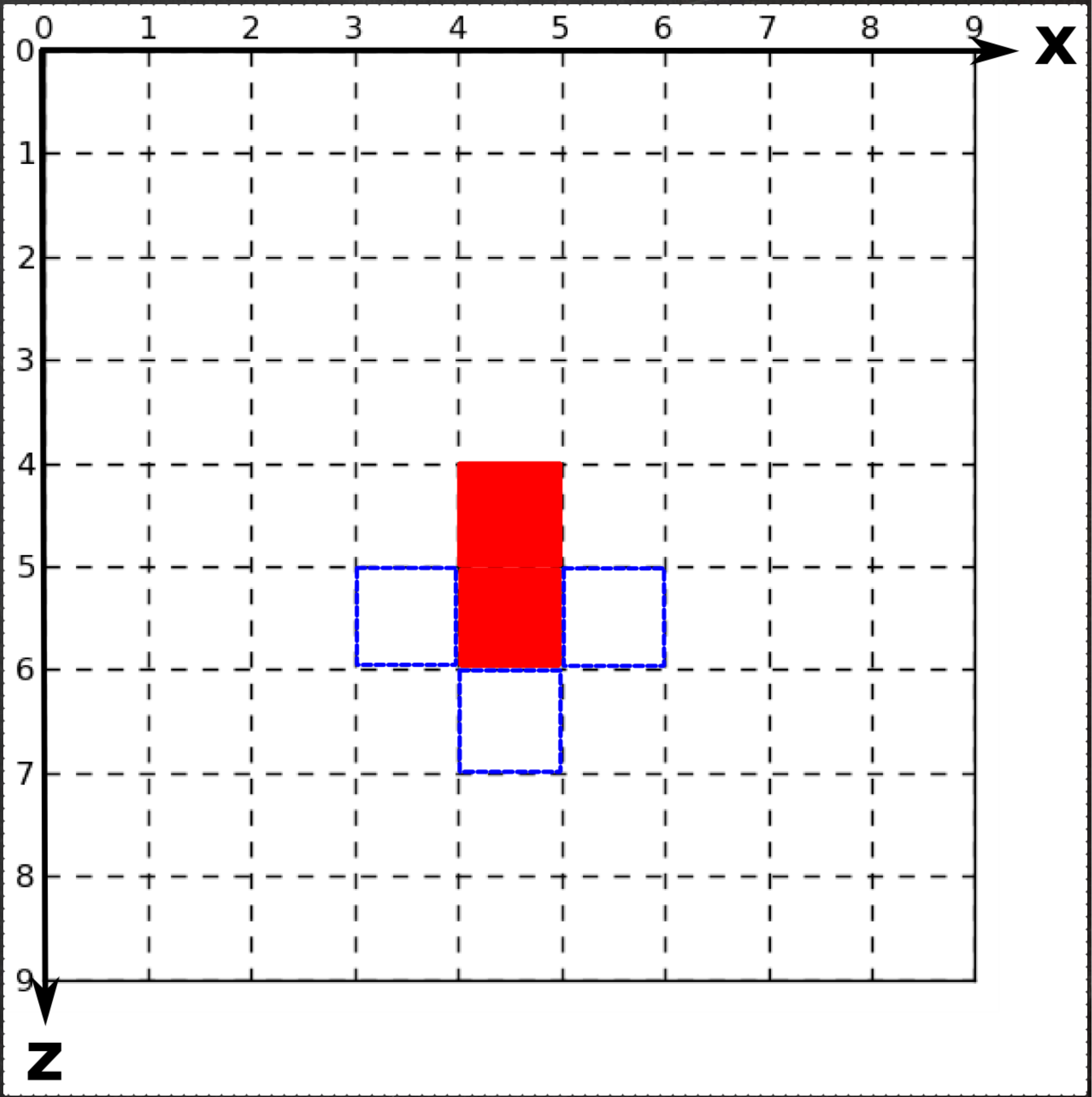
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



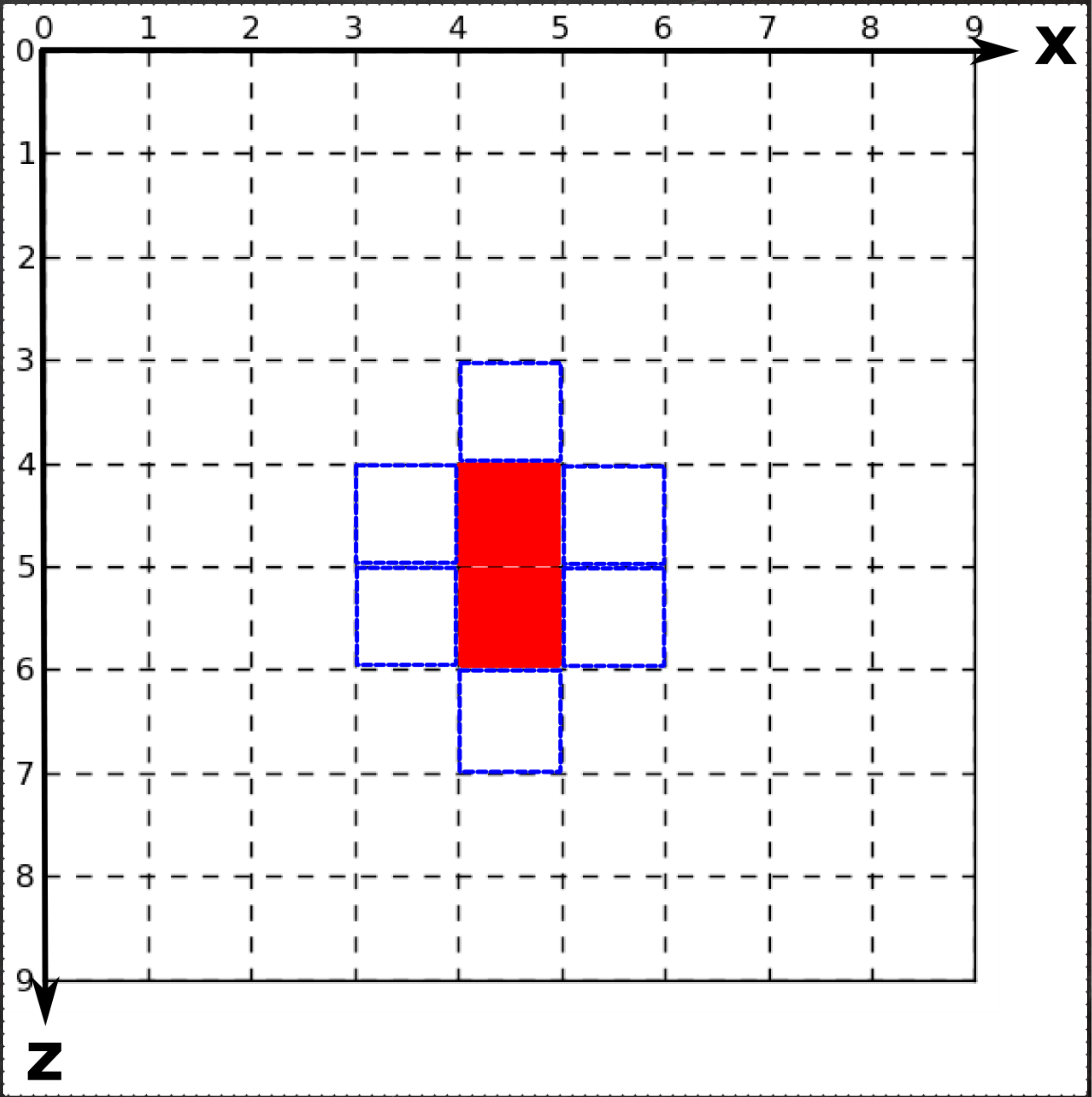
```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



```
r = d
for s in seeds:
  r -= dens[s]*Gtrans(s)
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans(n)
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```



Otimizações

```
r = d
for s in seeds:
  r -= dens[s]*Gtrans[s]
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

```
r = d
for s in seeds:
  r -= dens[s]*Gtrans[s]
  p[s] = dens[s]
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

- ✔ Crescer ⬇ misfit
- ✔ Crescer ⬆ reg

2 casos:

- ✔ misfit >> reg
  - ✔ Crescer ⬇ func. obj.
- ✔ Misfit << reg
  - ✔ Crescer ⬆ func. obj.
  - ✔ Não consegue crescer
  - ✔ Não ajusta os dados

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     if misfit + reg < best:
      best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

Para garantir o ajuste:

- ✔ Obrigatório ⬇ misfit
- ✔ Escolhe o que produz menor f. obj.

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
         best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

Para garantir o ajuste:

- ✔ Obrigatório ⬇ misfit
- ✔ Escolhe o que produz menor f. obj.
- ✔ Continua compacto
- ✔ ... e ajusta o dado

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

Só usa colunas dos vizinhos

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

... nunca mais usa a coluna best_n

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

- ✔ Não precisa calcular Jacobiana inteira

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

- Não precisa calcular Jacobiana inteira
- Só quando necessário (Lazy evaluation)

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
         best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

- ✔ Não precisa calcular Jacobiana inteira
- ✔ Só quando necessário (Lazy evaluation)
- ✔ Apaga coluna best_n

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

- ✔ Não precisa calcular Jacobiana inteira
- ✔ Só quando necessário (Lazy evaluation)
- ✔ Apaga coluna best_n
- ✔ Economiza RAM e processamento

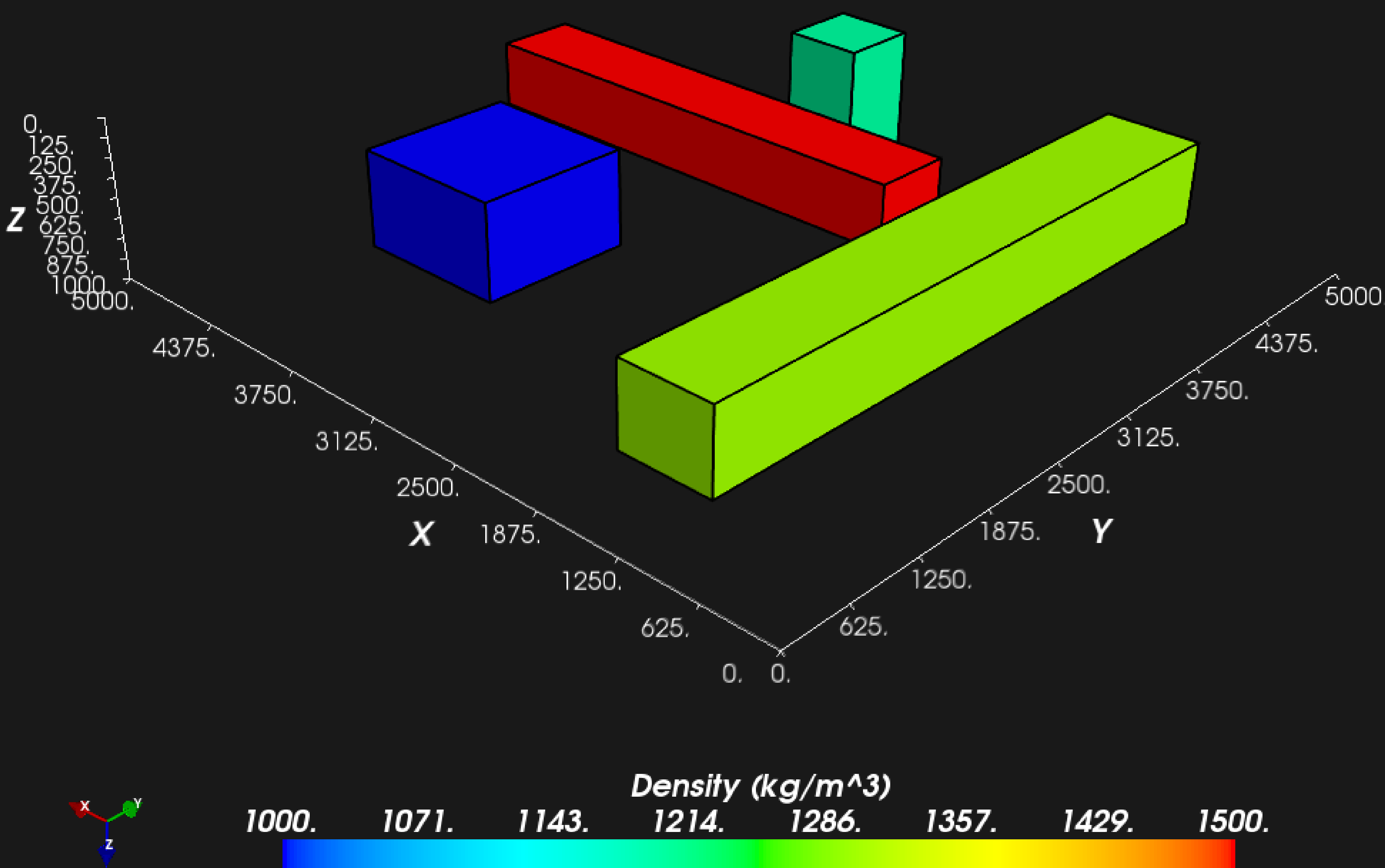
```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

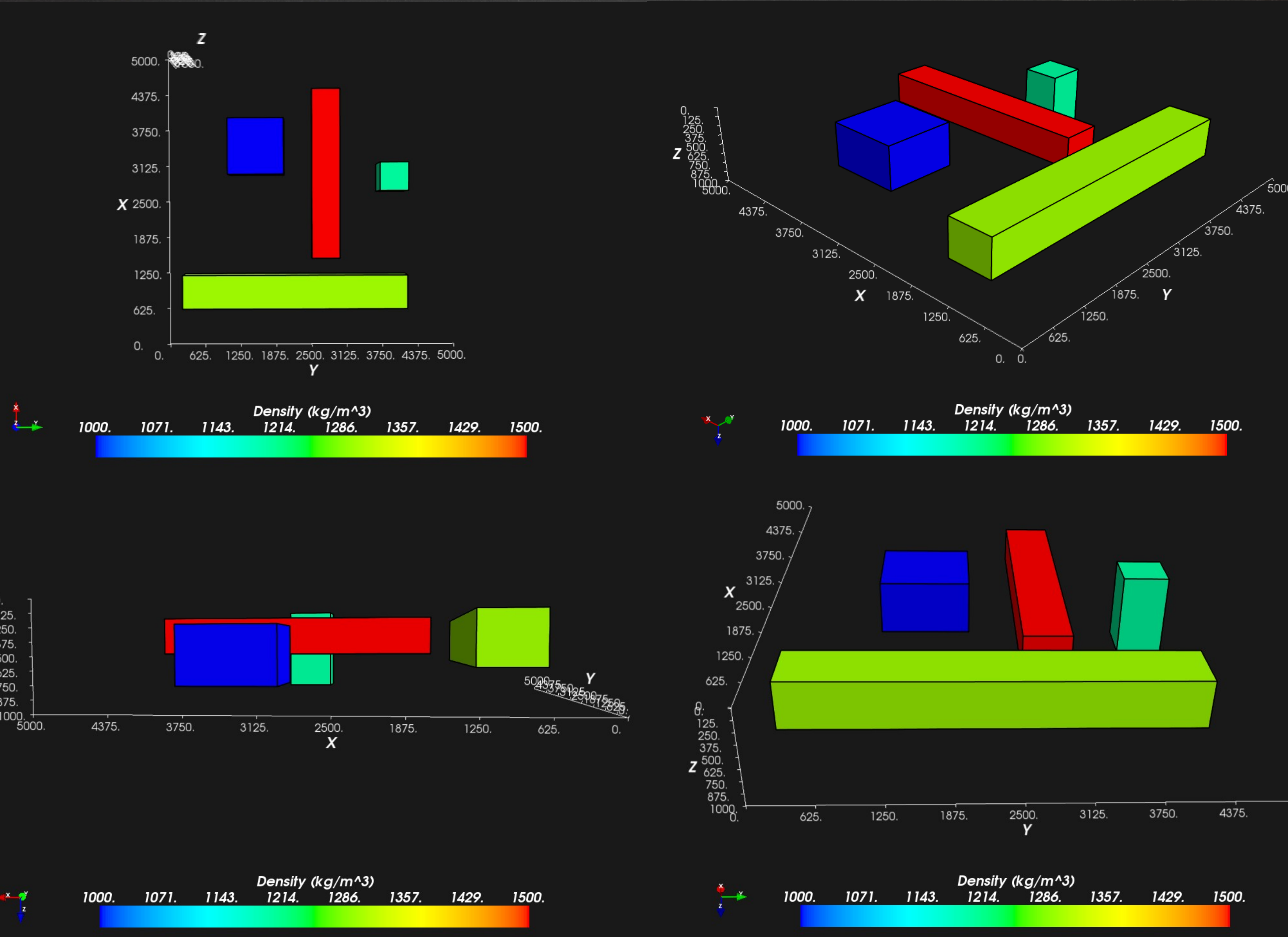
- ✔ Não precisa calcular Jacobiana inteira
- ✔ Só quando necessário (Lazy evaluation)
- ✔ Apaga coluna best_n
- ✔ Economiza RAM e processamento
- ✔ Inverter muitos dados

```
for i in maxit:
 for s in seeds:
   for n in neighbors[s]:
     p[n] = dens[s]
     newr -= p[n]*Gtrans[n]
     misfit = norm(newr)
     reg = regularizer(p)
     goal = misfit + reg
     if misfit < last_misfit:
       if goal < best_goal:
        best_n = n
   update_p_r(best_n)
   append_neighbors(best_n)
```

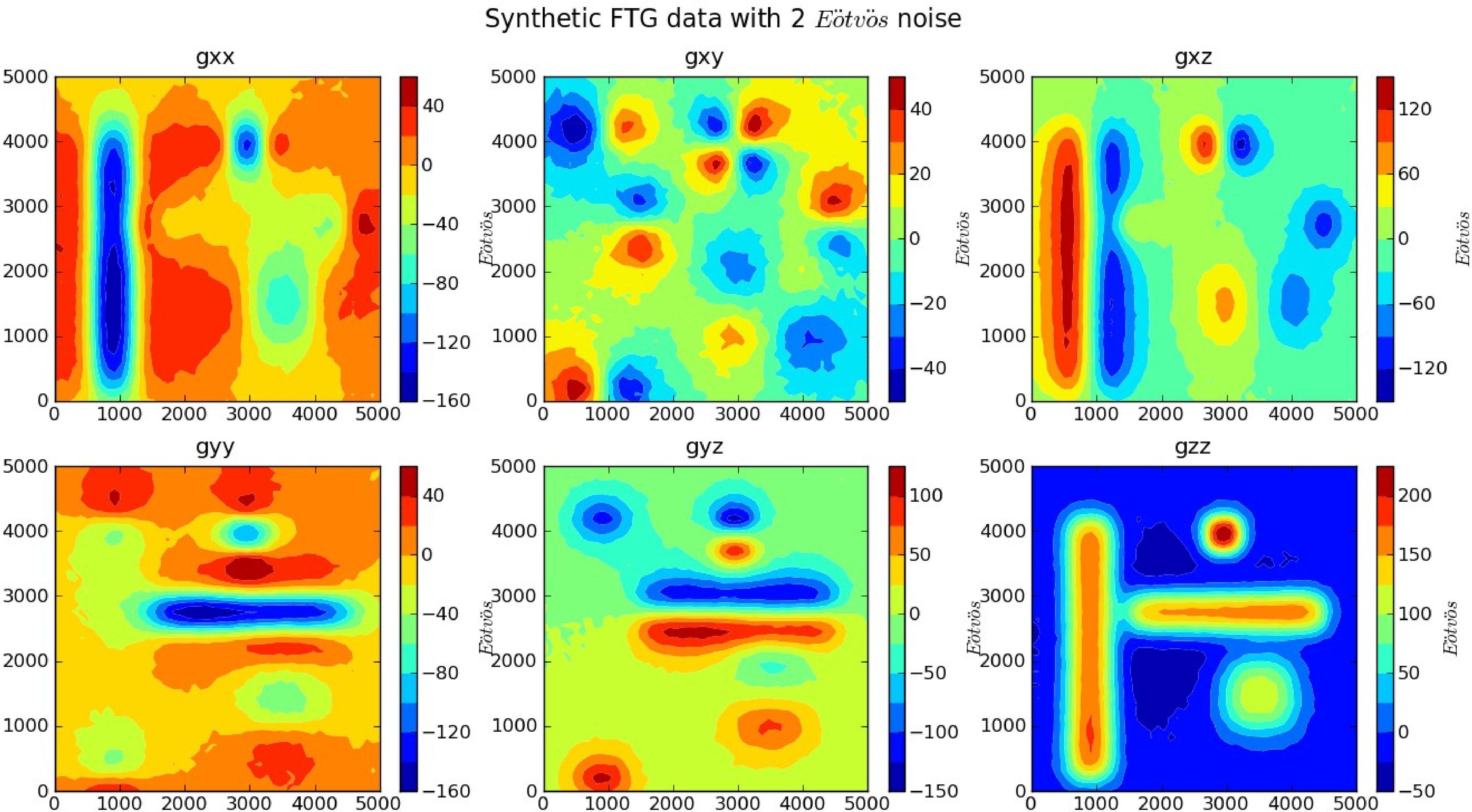
- ✔ Não precisa calcular Jacobiana inteira
- ✔ Só quando necessário (Lazy evaluation)
- ✔ Apaga coluna best_n
- ✔ Economiza RAM e processamento
- ✔ Inverter muitos dados
- ✔ … e com malhas finas

Dados sintéticos

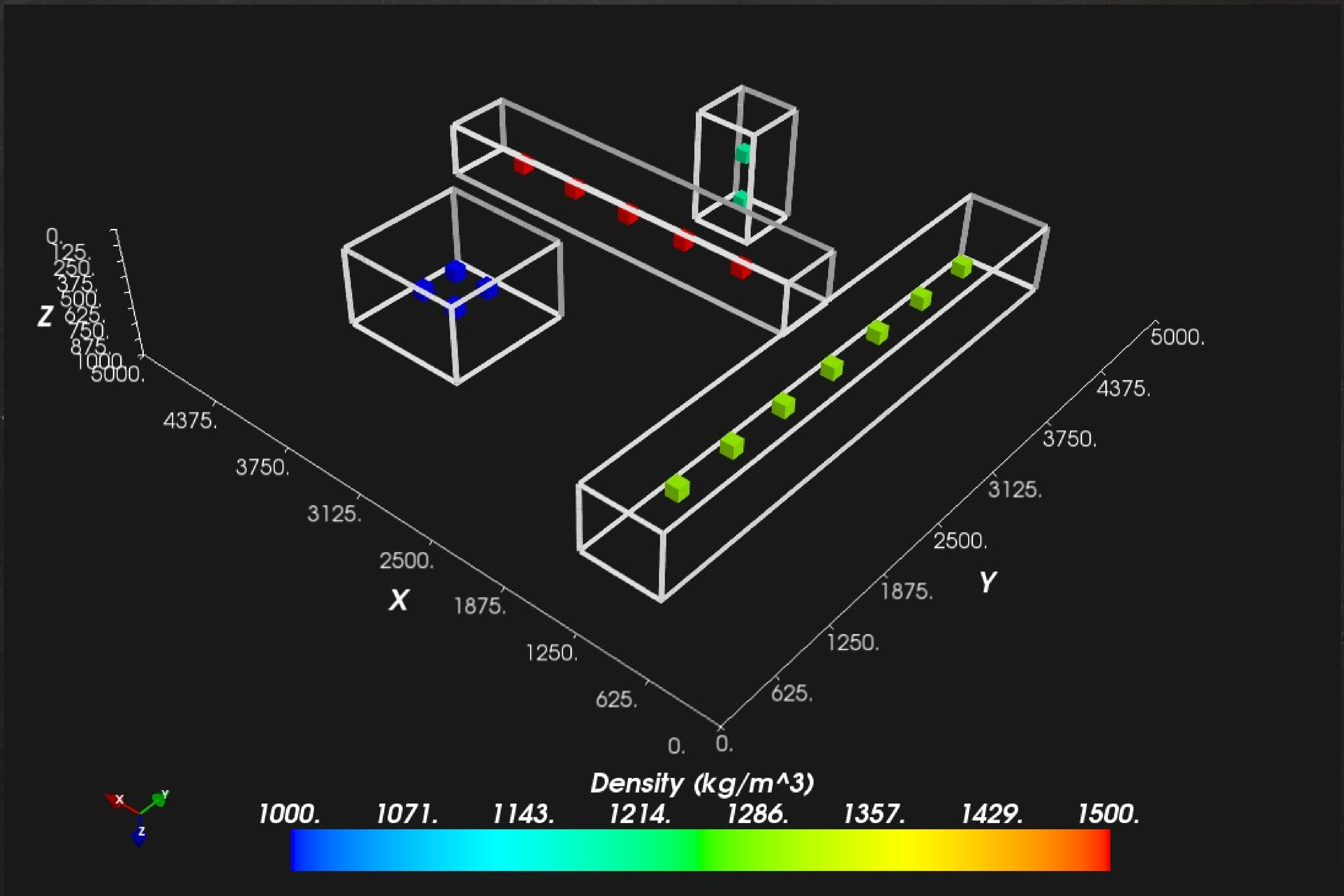

Z
0.
125.
250.
375.
500.
625.
750.
875.
1000.
5000.
4375.
3750.
3125.
2500.
X
1875.
1250.
625.
0.
0.
625.
1250.
1875.
Y
2500.
3125.
3750.
4375.
5000.
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
X
Y
Z



Z
5000.
4375.
3750.
3125.
X 2500.
1875.
1250.
625.
0.
0. 625. 1250. 1875. 2500. 3125. 3750. 4375. 5000.
Y
Density (kg/m^3)
1000. 1071. 1143. 1214. 1286. 1357. 1429. 1500.
0.
125.
250.
375.
Z 500.
625.
750.
875.
1000.
5000.
4375.
3750.
3125.
2500.
X
1875.
1250.
625.
0. 0.
625.
1250.
1875.
Y
2500.
3125.
3750.
4375.
Density (kg/m^3)
1000. 1071. 1143. 1214. 1286. 1357. 1429. 1500.
5000. 4375. 3750. 3125. 2500. 1875. 1250. 625. 0.
X
Y
Density (kg/m^3)
1000. 1071. 1143. 1214. 1286. 1357. 1429. 1500.
5000.
4375.
3750.
3125.
X
2500.
1875.
1250.
625.
0.
125.
250.
375.
Z 500.
625.
750.
875.
1000.
0.
625. 1250. 1875. 2500. 3125. 3750. 4375.
Y
Density (kg/m^3)
1000. 1071. 1143. 1214. 1286. 1357. 1429. 1500.

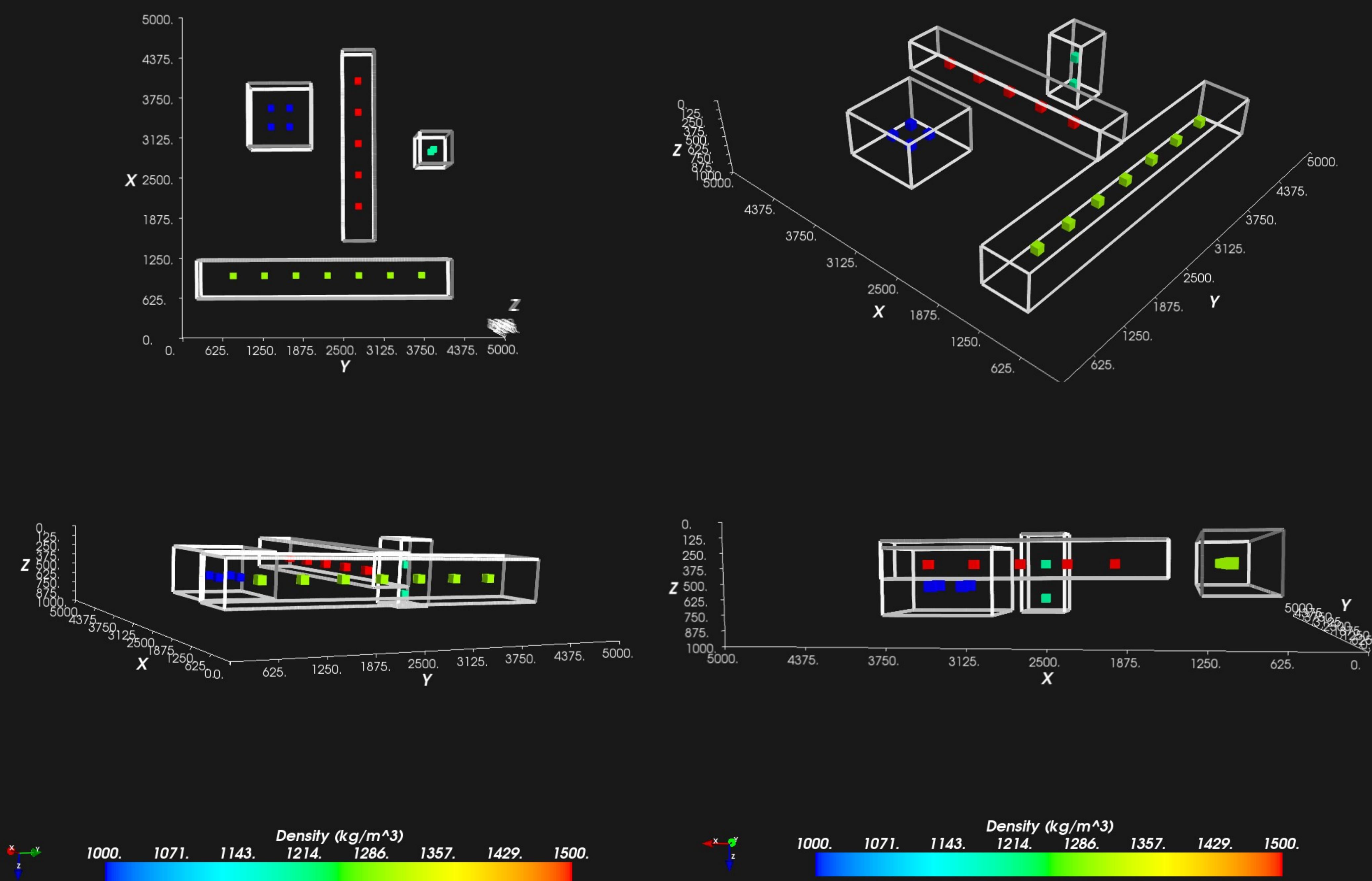

Synthetic FTG data with 2 Eötvös noise
gxx
gxy
gxz
gyy
gyz
gzz
Eötvös

Sementes

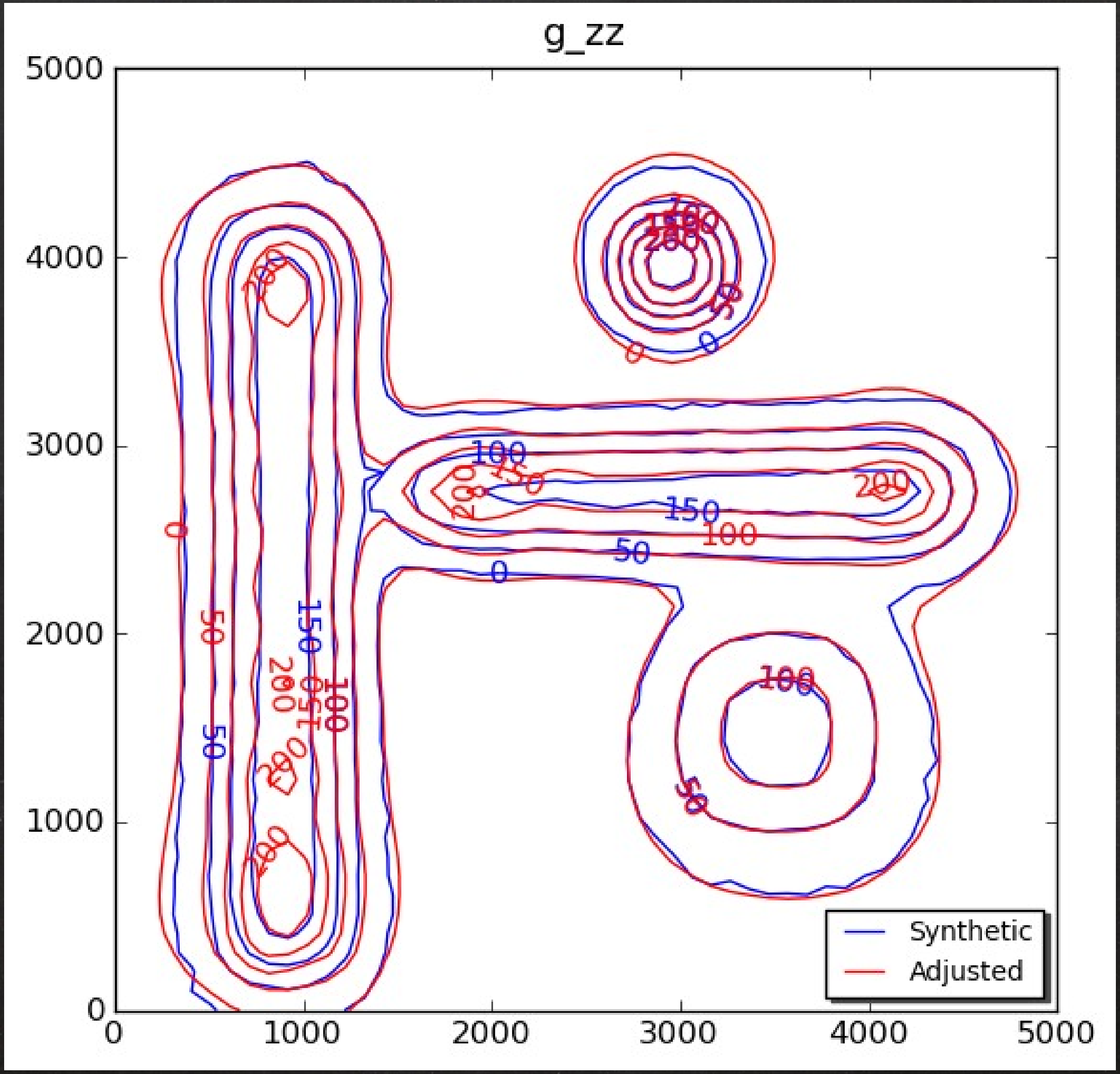


Z
0.
125.
250.
375.
500.
625.
750.
875.
1000.
X
5000.
4375.
3750.
3125.
2500.
1875.
1250.
625.
0.
Y
0.
625.
1250.
1875.
2500.
3125.
3750.
4375.
5000.
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.

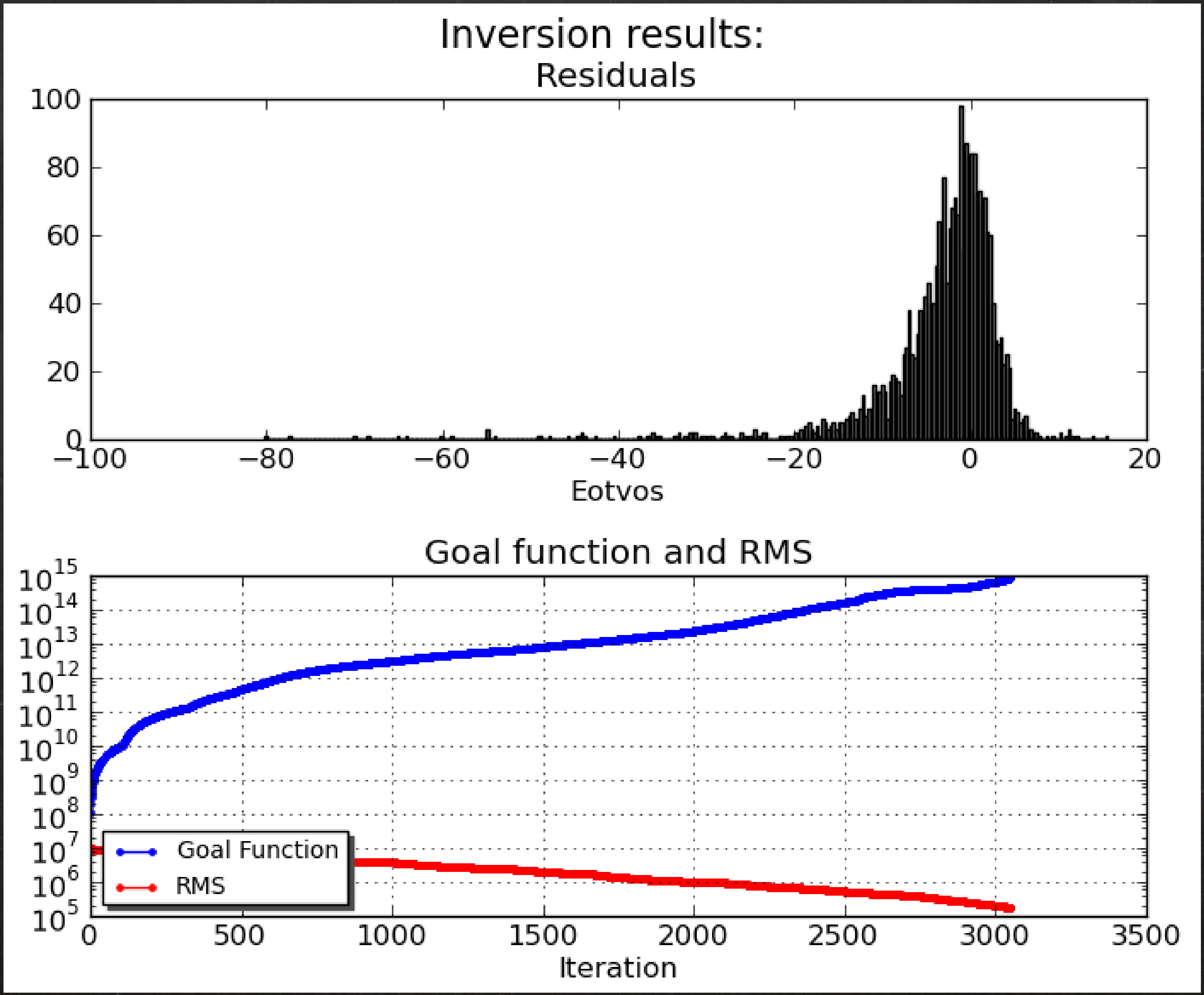
Sementes



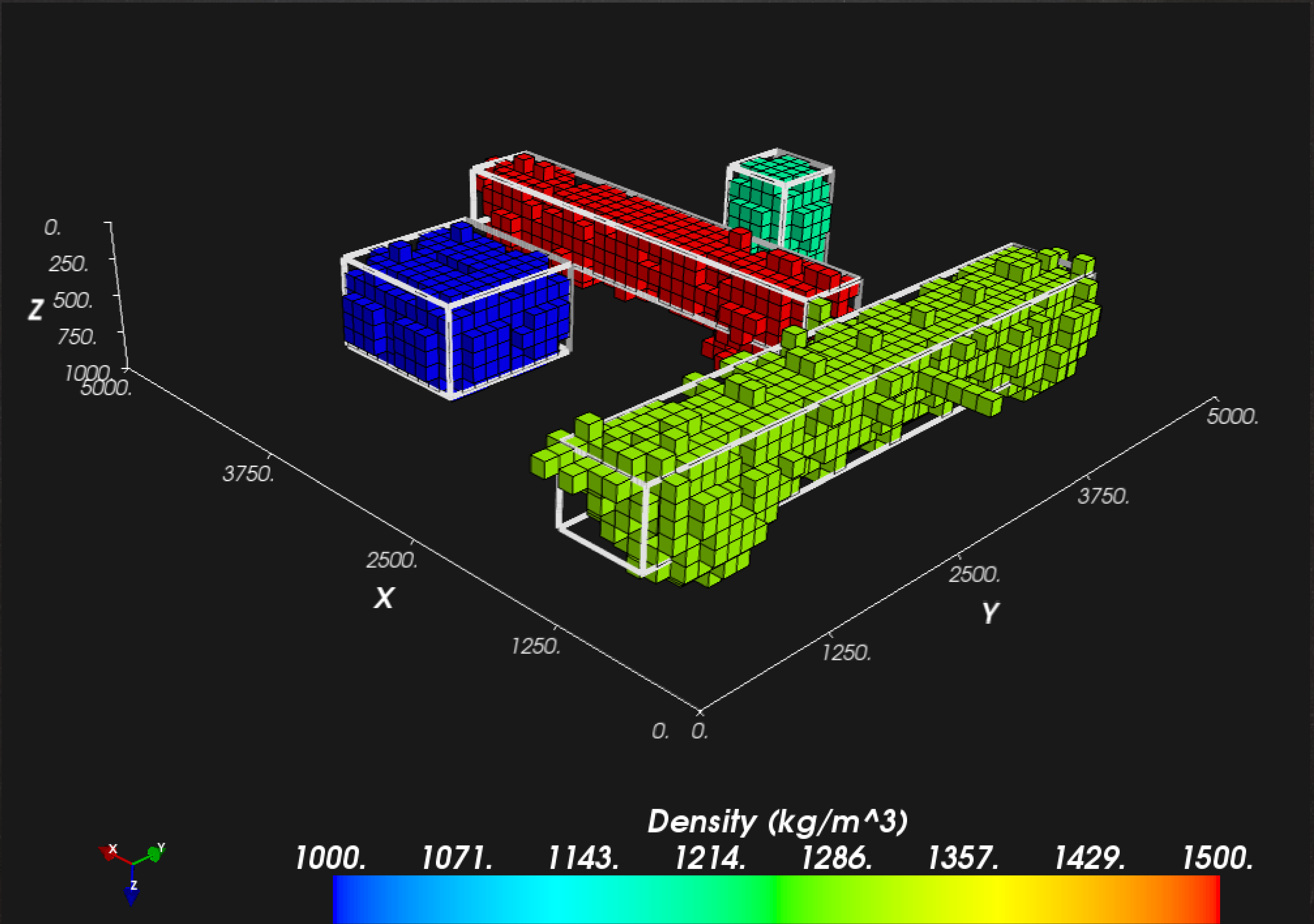
Resultados



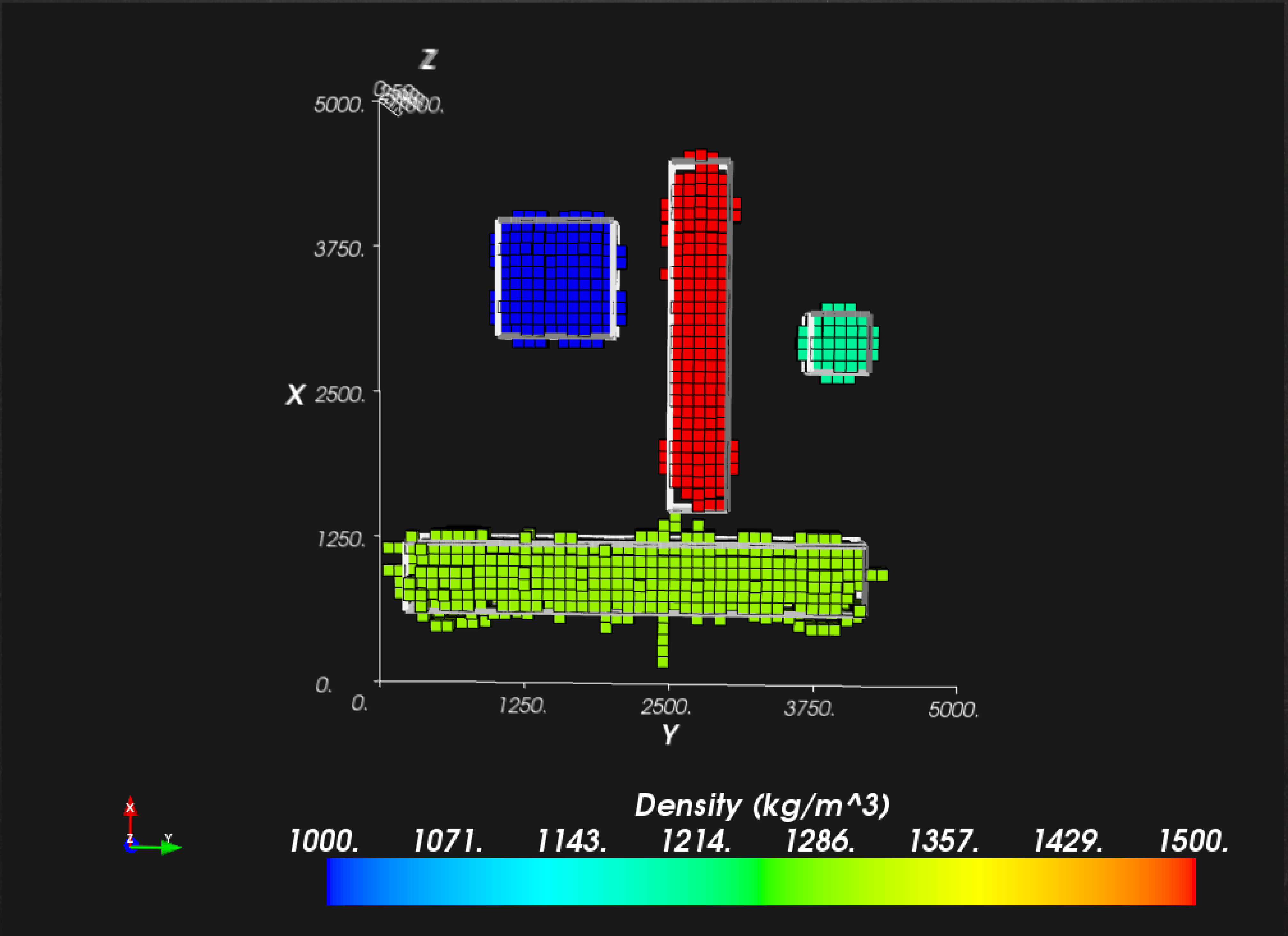
Resultados

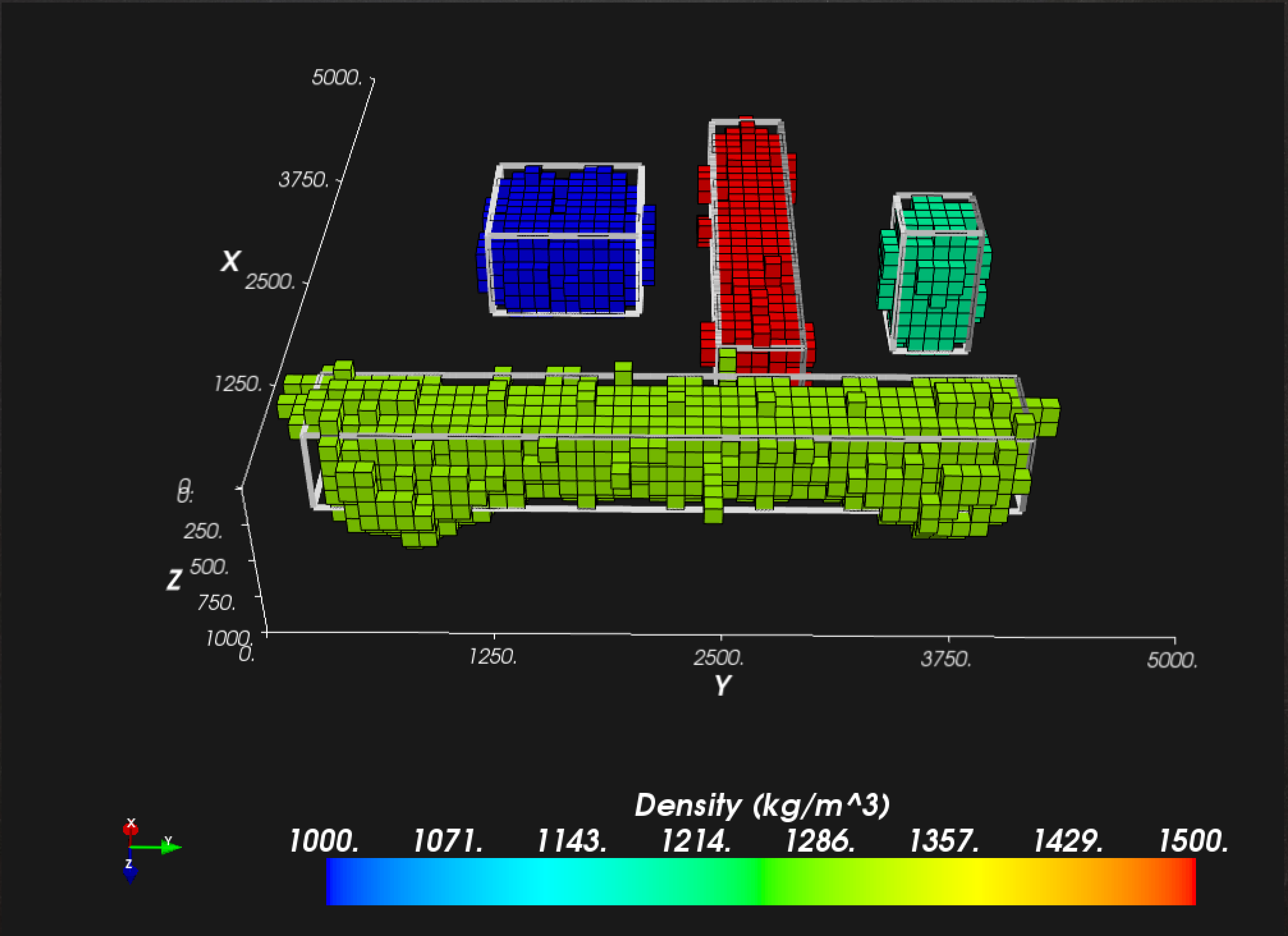


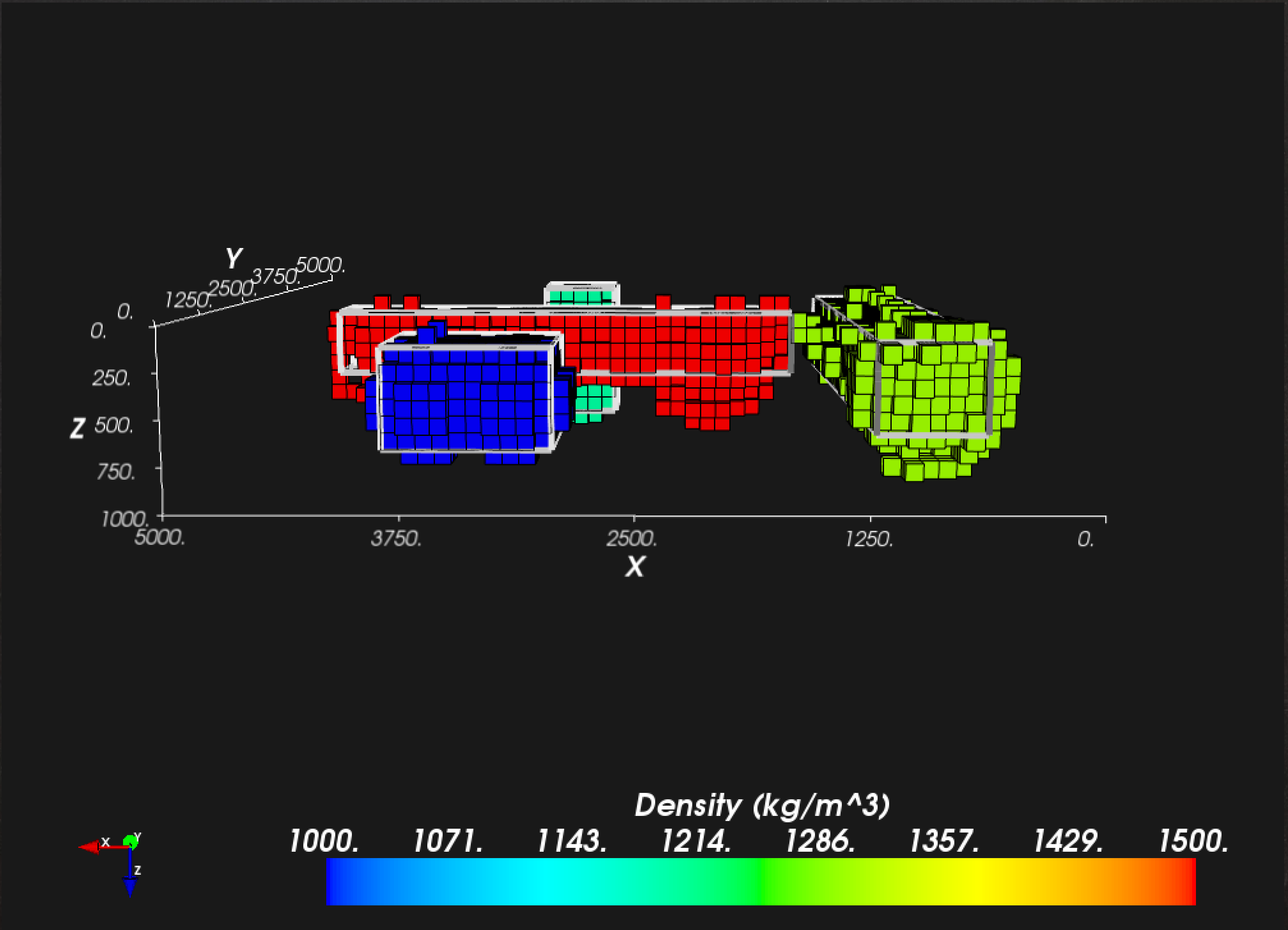
Inversion results:
Residuals
100
80
60
40
20
0
−100
−80
−60
−40
−20
0
20
Eotvos
Goal function and RMS
10^{15}
10^{14}
10^{13}
10^{12}
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
Goal Function
RMS
0
500
1000
1500
2000
2500
3000
3500
Iteration



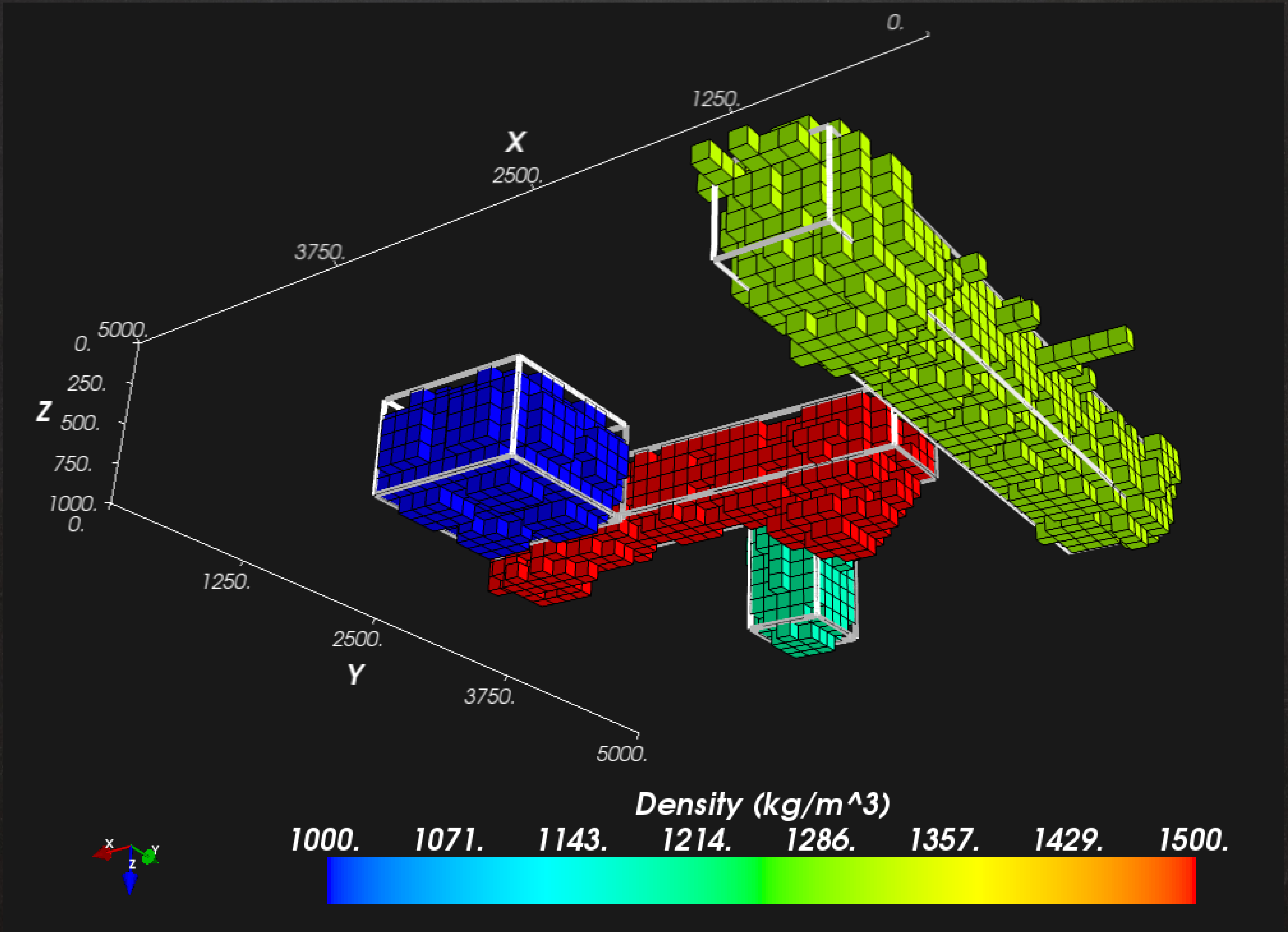
0.
250.
Z
500.
750.
1000.
5000.
3750.
2500.
X
1250.
0.
0.
1250.
2500.
Y
3750.
5000.
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
X
Y
Z


Z
5000.
3750.
X 2500.
1250.
0.
0.
1250.
2500.
3750.
5000.
Y
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
X
Z
Y

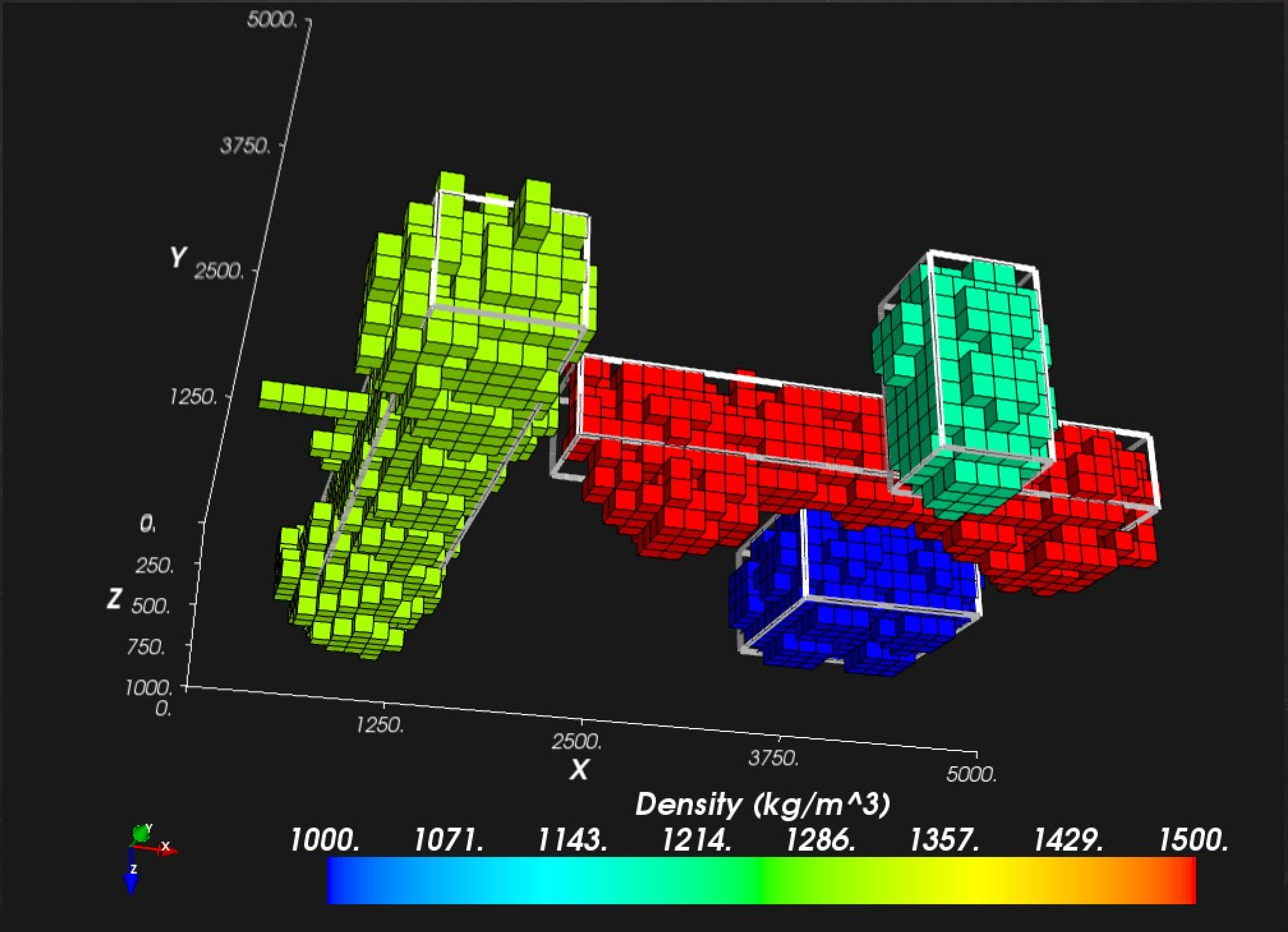

5000.
3750.
X
2500.
1250.
0.
250.
Z
500.
750.
1000.
0.
1250.
2500.
3750.
5000.
Y
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
X
Y
Z

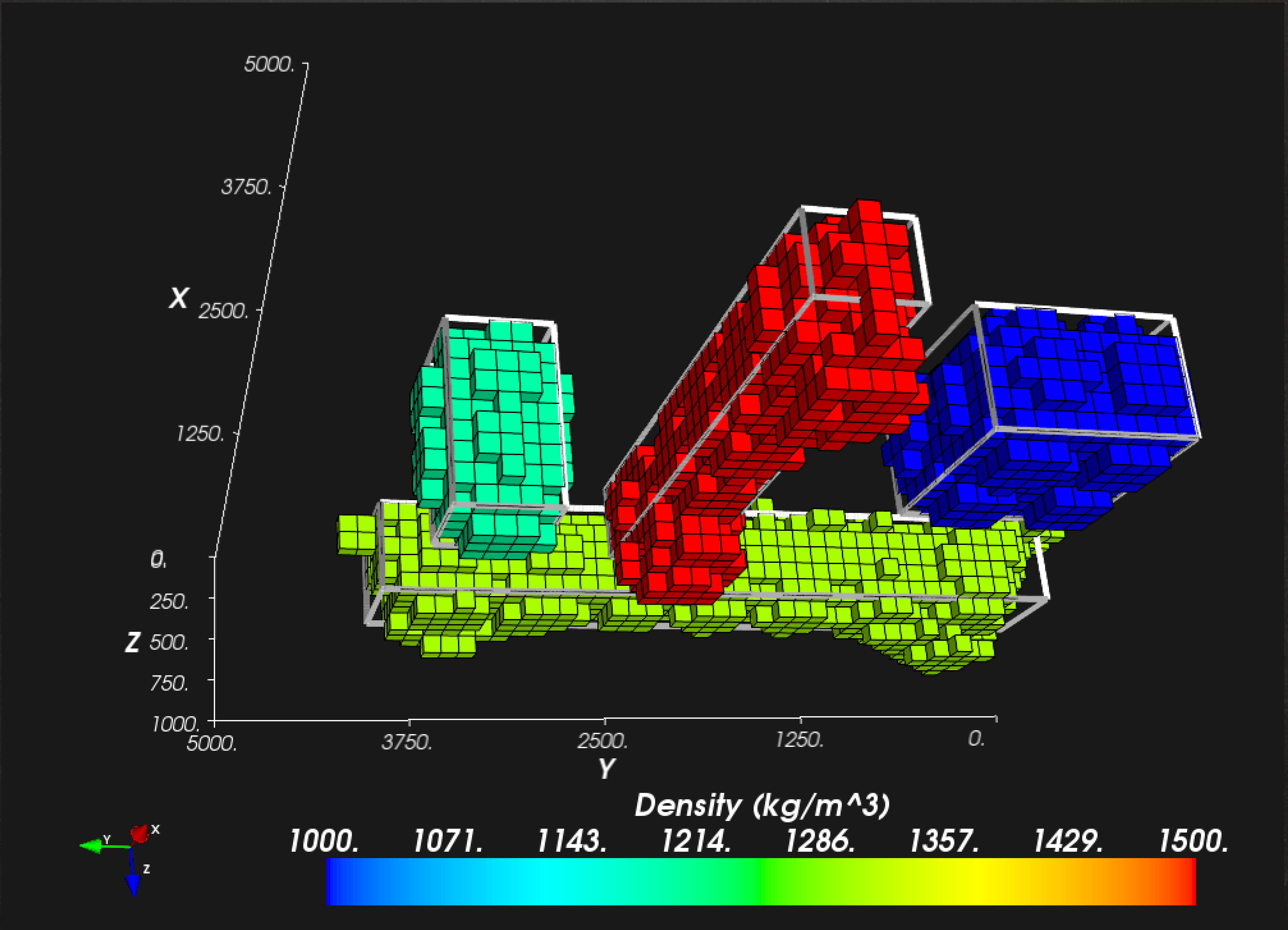


Y
1250.
2500.
3750.
5000.
0.
0.
250.
Z 500.
750.
1000.
5000.
3750.
2500.
1250.
0.
X
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
x
y
z



X
0.
1250.
2500.
3750.
5000.
Z
0.
250.
500.
750.
1000.
Y
0.
1250.
2500.
3750.
5000.
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.


5000.
3750.
Y
2500.
1250.
0.
250.
Z
500.
750.
1000.
0.
1250.
2500.
3750.
5000.
X
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.



5000.
3750.
X
2500.
1250.
0.
250.
Z 500.
750.
1000.
5000.
3750.
2500.
1250.
0.
Y
Density (kg/m^3)
1000.
1071.
1143.
1214.
1286.
1357.
1429.
1500.
Y
X
Z