GPU Isosurface Raycasting of Volume Datasets Based On Box-Splines

Minho Kim
School of Computer Science
University of Seoul

Introduction

Related Articles

- **[Kim 2012]** Minho Kim "*GPU Isosurface Raycasting of FCC Datasets*," Graphical Models, 2012 (in print)

- **[Kim & Lee 2012]** Minho Kim & Young-joon Lee, "*Real-time BCC Volume Isosurface Ray Casting on the GPU*," Journal of the Korea Computer Graphics Society, Dec. 2012

Objectives

- Fast spline evaluation on the GPU
  - How to ditch conditional branches & lookup tables?
  - How to minimize data fetch overhead?
  - How to further reduce computational overheads?
- Fast & accurate normal computation
- Further performance improvement

GPU Raycasting

- [Sigg & Hadwiger 2005] tricubic B-spline on the Cartesian lattice
- [Csébfalvi & Hadwiger 2006] tricubic B-spline on the BCC lattice
- [Finkbeiner et al. 2010] 8-direction quintic box-spline on the BCC lattice
- **[Kim 2012] 6-direction cubic box spline on the FCC lattice**
- **[Kim & Lee 2012] 7-direction quartic box-spline on the BCC lattice**

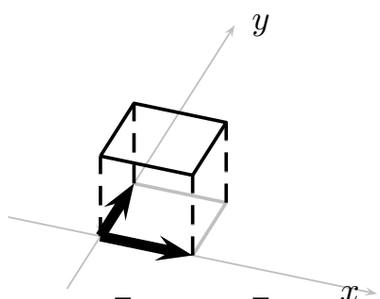
Results

- For FCC (Face-Centered Cubic) datasets [Kim 2012]
  - x2.4 than BCC [Finkbeiner et al. 2010]
  - x0.4 than the Cartesian [Sigg & Hadwiger 2005]
- For BCC (Body-Centered Cubic) datasets [Kim & Lee 2012]
  - x1.2 than [Finkbeiner et al. 2010]
  - x0.4 than [Csébfalvi and Hadwiger 2006]

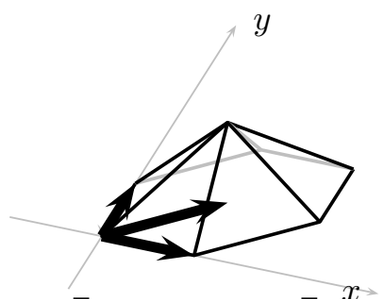
Background

Box-Splines

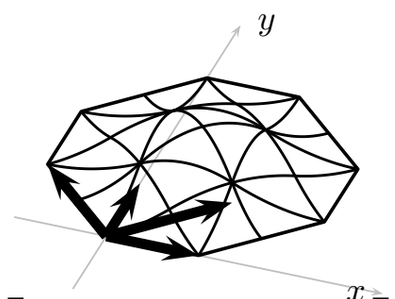
- Defined by an $n \times m$ "direction matrix" Ξ
  - Directional convolution for each direction



$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

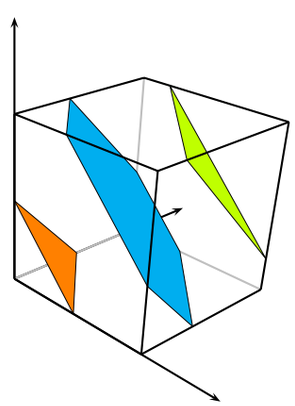


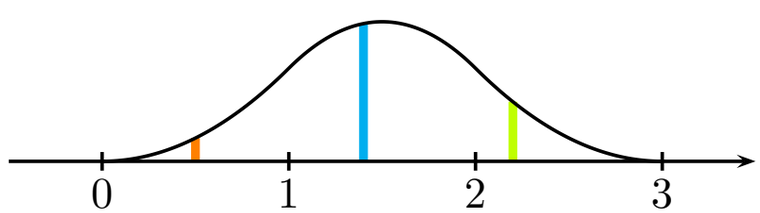
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

  - Projected volume density





Box-Splines

- Piecewise polynomial of degree = $m - n$
- Polynomial pieces are delineated by ***knot planes***, hyperplanes defined by the directions
- Finite support defined by Minkowski sum of the directions

Box-Splines

- Approximation order = $\rho(\Xi)$
  - $\rho(\Xi)$: minimum # of directions in Ξ to make remaining directions have rank $< n$
    - Given the degree, we get better approximation order if the directions are more “spread out.”
- Quasi-interpolator required to obtain the maximal approximation order

Box-Splines

- Box-splines on non-Cartesian lattices
  - Change-of-variables

$$\sum_{\boldsymbol{j}\in\mathbf{G}\mathbb{Z}^n} |\det \mathbf{G}| M_{\mathbf{G}\Xi}(\cdot - \boldsymbol{j}) f(\boldsymbol{j}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} M_{\Xi}(\mathbf{G}^{-1} \cdot - \boldsymbol{k}) f(\mathbf{G}\boldsymbol{k})$$

  - [Kim & Peters 2010]

    “Symmetric Box-Spline on the $\mathcal{A}_n{}^*$ Lattice”

  - [Kim & Peters 2011]

    “Symmetric Box-Splines on Root Lattices”

Box-Splines vs. B-Splines

	Box-Splines	B-Splines
approximation order	High	Low
non-Cartesian lattices	Yes	(Mostly) No
evaluation	Complicated	de Boor's

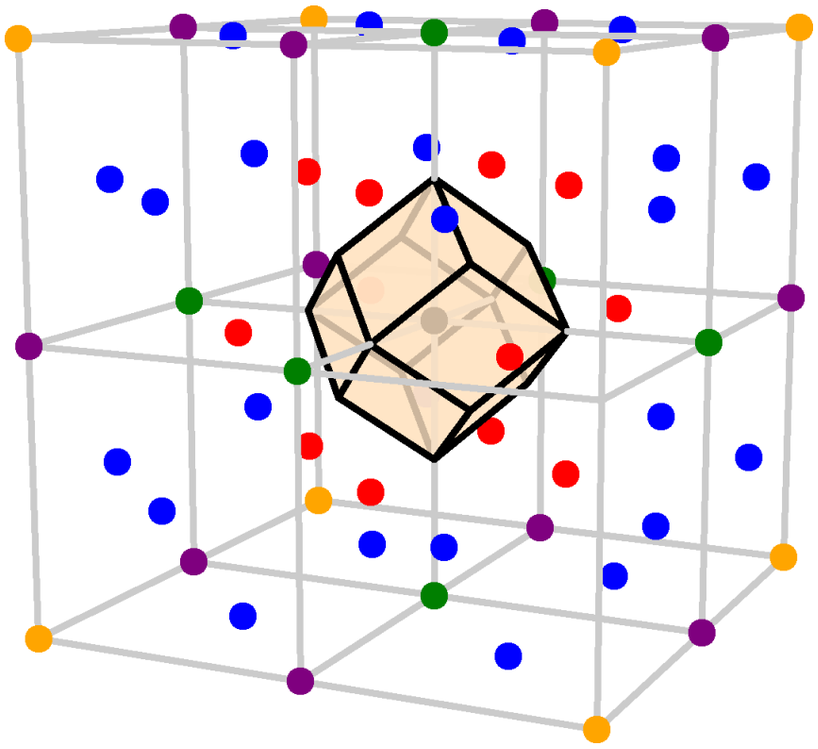
C. de Boor
K. Höllig
S. Riemenschneider

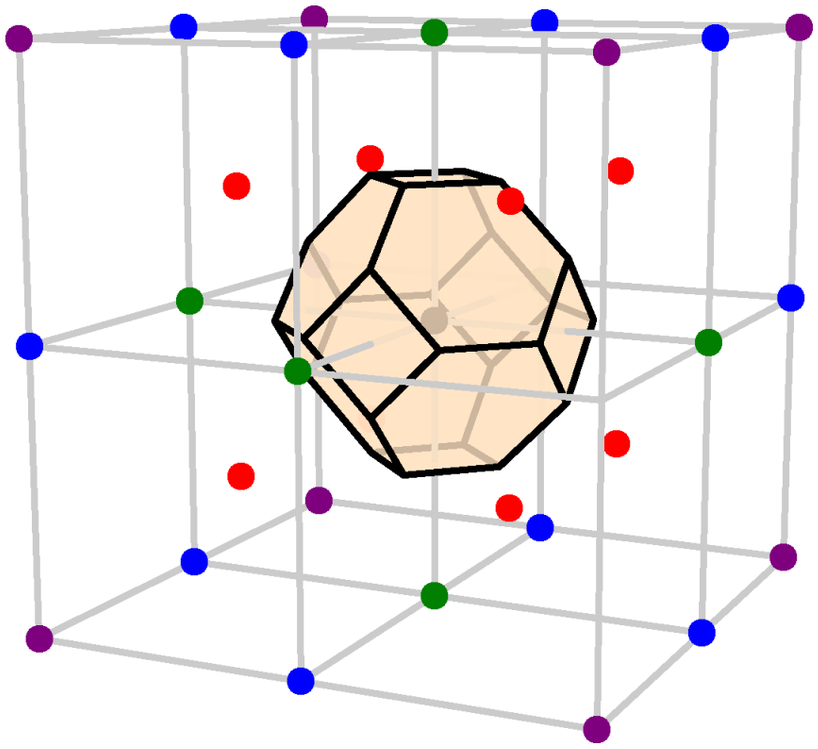
Applied Mathematical Sciences 98

Box Splines

Springer-Verlag

The FCC & BCC Lattices





$$\mathbb{Z}_{\text{fcc}} := \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \mathbb{Z}^3 \qquad \mathbb{Z}_{\text{bcc}} := \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \mathbb{Z}^3$$

The FCC & BCC Lattices

- More efficient sampling lattices than the Cartesian lattice
  - Requires less # of samples to reconstruct an isotropic and band-limited signal
- Symmetric box-splines are good candidates as reconstruction filters
  - BCC: [Entezari et al. 2004] [Csébfalvi & Hadwiger 2006] [Kim 2012]
  - FCC: [Kim et al. 2008]

Efficient Evaluation

Deriving Symbolic Formula

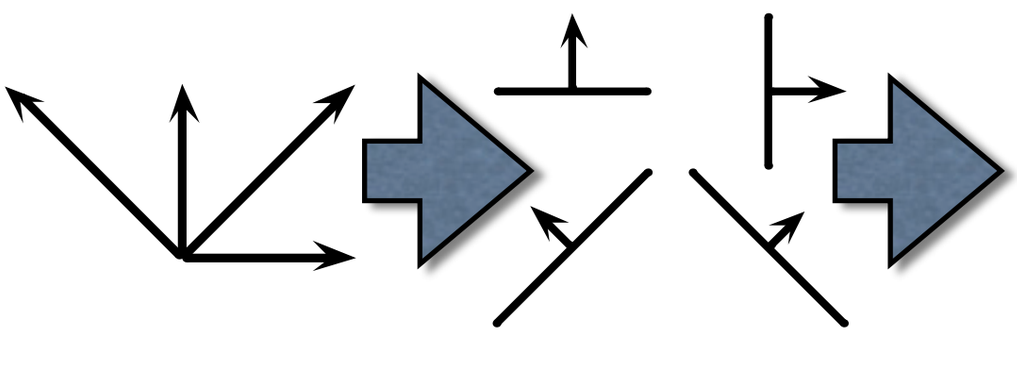
directions

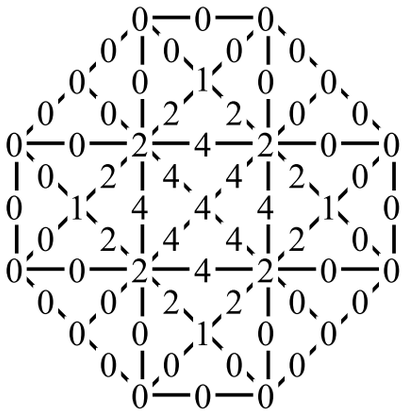
knot planes

spline structure

shift-invariant polynomial pieces

box-spline





BB-coefficients

[Kim & Peters 09]

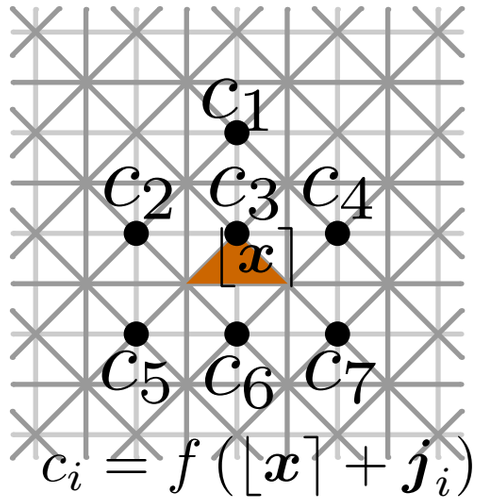
$$b_{200} = c_1 + c_2 + 4c_3 + c_4 + c_6$$

$$b_{110} = 2c_2 + 4c_3 + 2c_6 \qquad b_{101} = 4c_3 + 2c_4 + 2c_6$$

$$b_{011} = 4c_3 + 4c_6$$

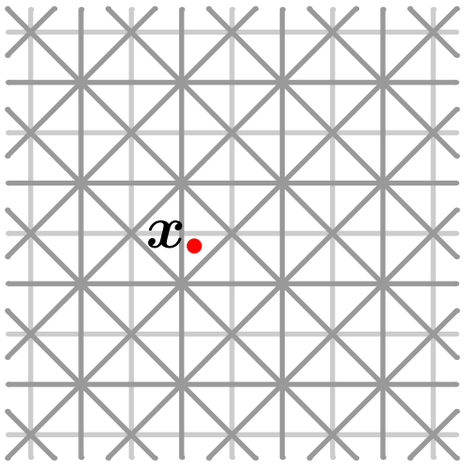
$$b_{020} = 2c_2 + 2c_3 + 2c_5 + 2c_6 \qquad b_{002} = 2c_3 + 2c_4 + 2c_6 + 2c_7$$

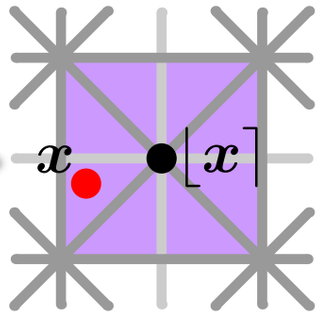
symbolic formula

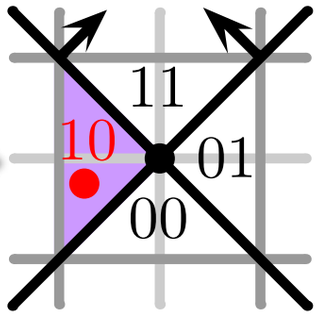


stencil

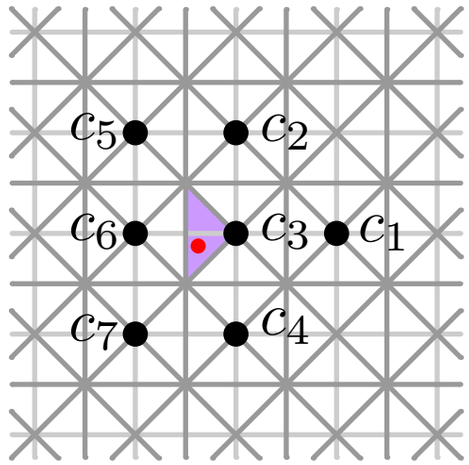
Evaluation





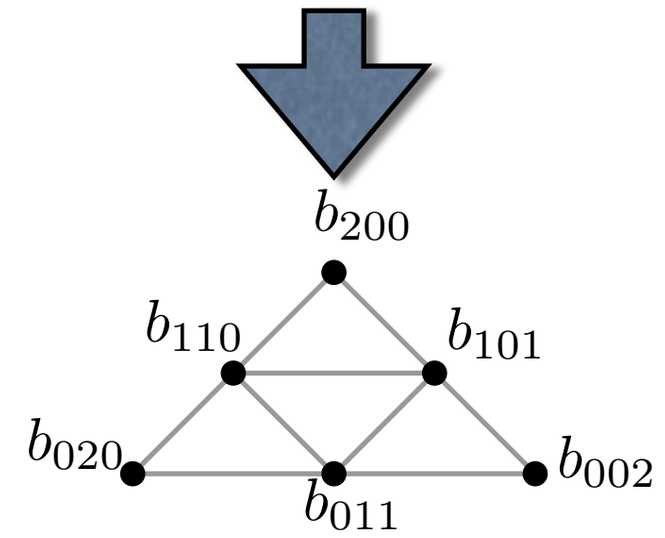


$$T_{10} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



$$c_i = f\left(\lfloor \boldsymbol{x} \rceil + T_{10}^{-1} \boldsymbol{j}_i\right)$$

function EVALUATEQUADRATIC(f, $\boldsymbol{x}$)
  $\dot{\boldsymbol{x}} \leftarrow \boldsymbol{x} - \lfloor \boldsymbol{x} \rceil$
  $\boldsymbol{\tau} \leftarrow U\left((-1,1)\cdot\dot{\boldsymbol{x}}, (1,1)\cdot\dot{\boldsymbol{x}}\right)$
  Find $T_{\boldsymbol{\tau}}$
  Conditional branches & lookup tables
  for $i = 1$ to 7 **do**
    $c_i \leftarrow f(\lfloor \boldsymbol{x} \rceil + T_{\boldsymbol{\tau}}^{-1} \boldsymbol{j}_i)$ Matrix-vector multiplication
  end for
  $\boldsymbol{u} \leftarrow BT_{\boldsymbol{\tau}} \begin{bmatrix} \dot{\boldsymbol{x}} \\ 1 \end{bmatrix}$ (Optional)
  Construct the BB-form or other formula and evaluate it.
end function



$$\boldsymbol{u} = BT_{10} \begin{bmatrix} \boldsymbol{x} - \lfloor \boldsymbol{x} \rceil \\ 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & -1/2 & 1/2 \\ 0 & -1/2 & -1/2 \\ 1 & 1 & 1 \end{bmatrix}^{-1}$$

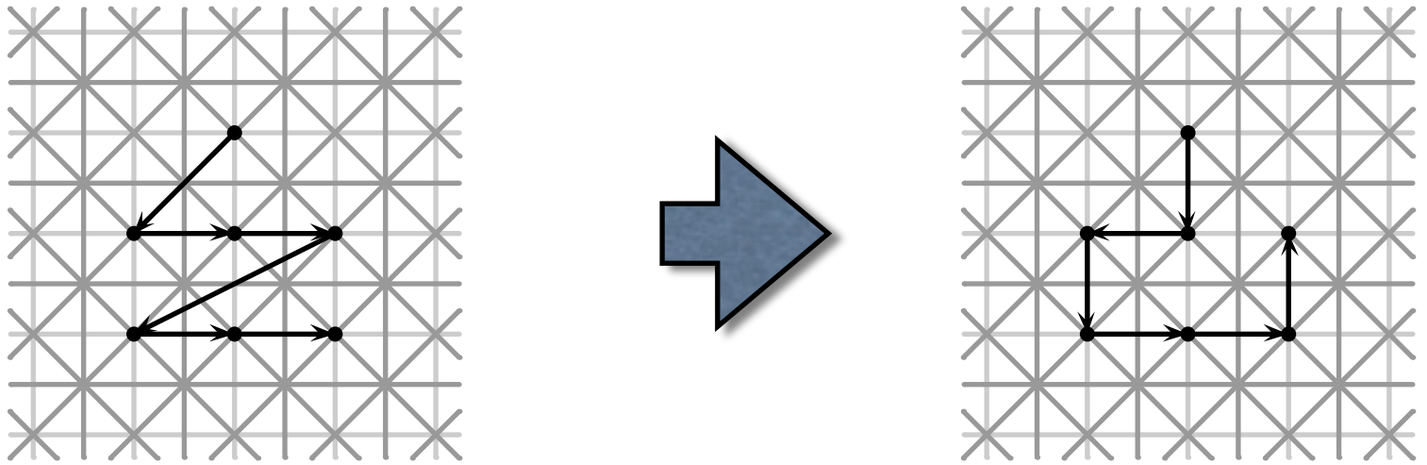
Encoding Matrix

$$T_{00} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad T_{01} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$T_{10} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad T_{11} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$T_{\boldsymbol{\tau}} = \begin{bmatrix} !(\boldsymbol{\tau}_1 \& \boldsymbol{\tau}_2) & (\boldsymbol{\tau}_1 \& \boldsymbol{\tau}_2)(1 - 2\boldsymbol{\tau}_2) \\ (\boldsymbol{\tau}_1 \& \boldsymbol{\tau}_2)(1 - 2\boldsymbol{\tau}_2) & !(\boldsymbol{\tau}_1 \& \boldsymbol{\tau}_2)(1 - 2\boldsymbol{\tau}_2) \end{bmatrix}$$

Axis-Aligned Fetching



$$c_i = f\left(\lfloor \boldsymbol{x} \rceil + T_{\boldsymbol{\tau}}^{-1} \boldsymbol{j}_i\right)$$

Matrix-vector multiplication

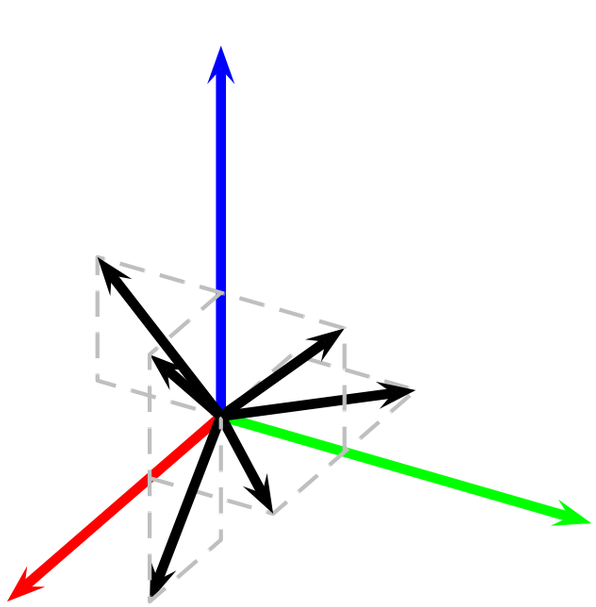
$$\begin{aligned}
\boldsymbol{y} &= \lfloor \boldsymbol{x} \rceil + (0, -1) \\
c_1 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} - T_{\boldsymbol{\tau}}^{-1}(:, 2) \\
c_3 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} - T_{\boldsymbol{\tau}}^{-1}(:, 1) \\
c_2 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} - T_{\boldsymbol{\tau}}^{-1}(:, 2) \\
c_5 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} + T_{\boldsymbol{\tau}}^{-1}(:, 1) \\
c_6 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} + T_{\boldsymbol{\tau}}^{-1}(:, 1) \\
c_7 &= f(\boldsymbol{y}) \\
\boldsymbol{y} &= \boldsymbol{y} + T_{\boldsymbol{\tau}}^{-1}(:, 2) \\
c_4 &= f(\boldsymbol{y})
\end{aligned}$$

Vector-vector addition

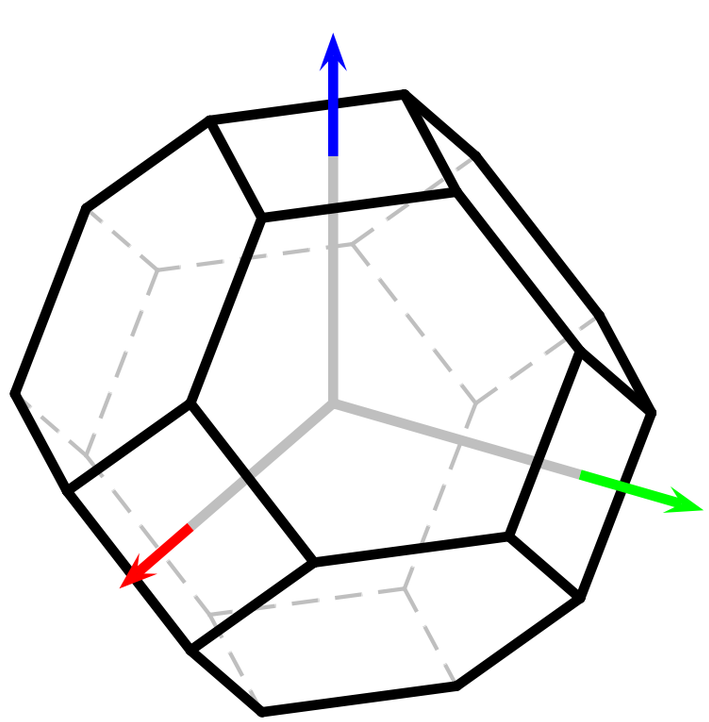
FCC Datasets

6-Direction Cubic Box-Spline on the FCC Lattice

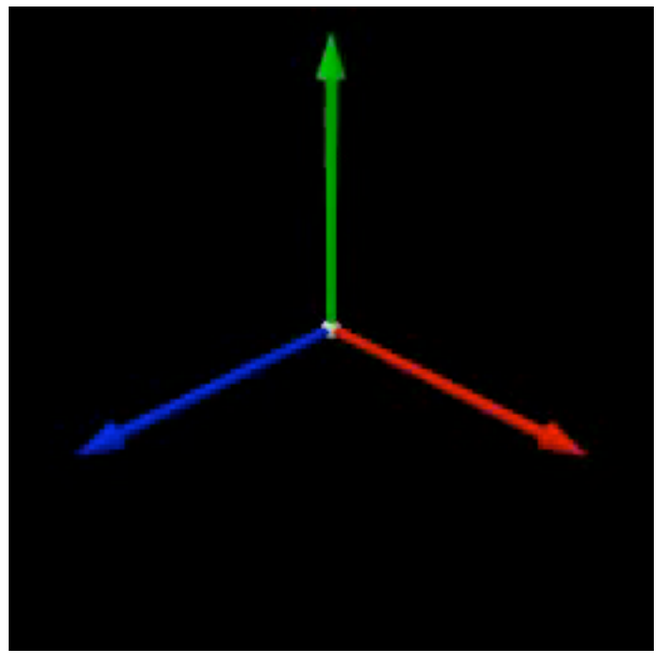
- [Entezari 2007] [Kim et al. 2008]
- Approximation order is 3



directions

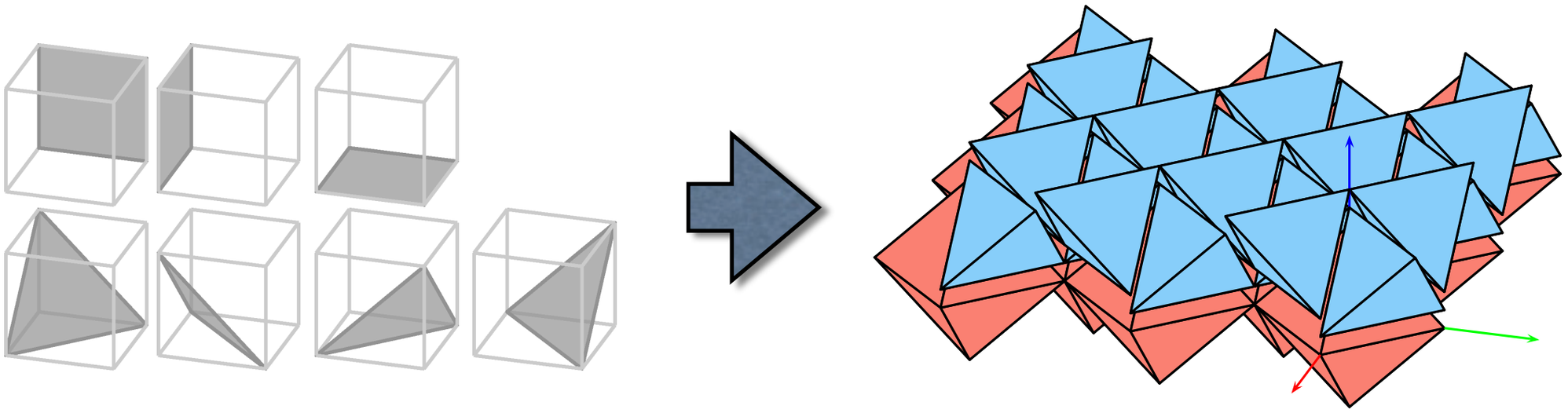


support



level sets

Polynomial Structure



$\lfloor \boldsymbol{x} \rfloor \in \mathbb{Z}_{\text{fcc}}$

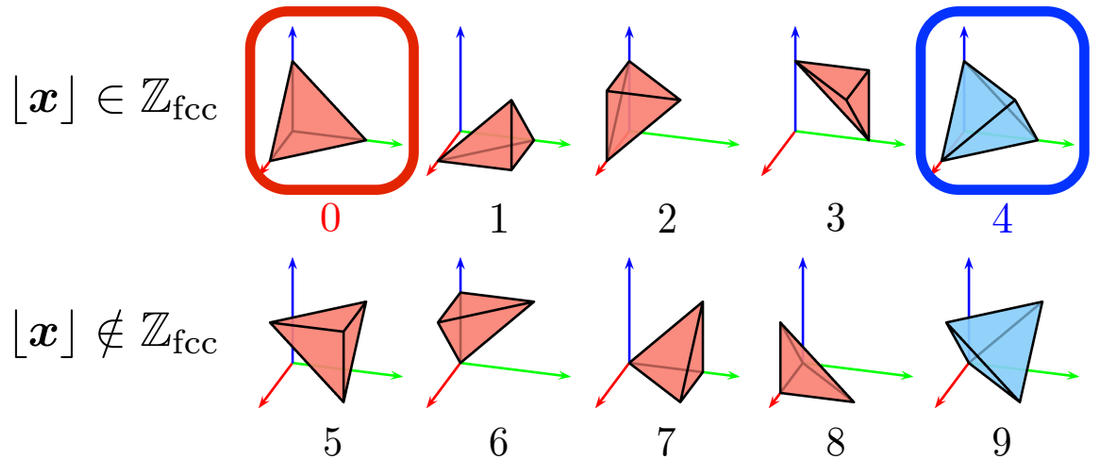
0 1 2 3 4

$\lfloor \boldsymbol{x} \rfloor \notin \mathbb{Z}_{\text{fcc}}$

5 6 7 8 9

Finding Poly Index

$\dot{\boldsymbol{x}} \leftarrow \boldsymbol{x} - \lfloor \boldsymbol{x} \rfloor$
$\pi \leftarrow ((1,1,1) \cdot \lfloor \boldsymbol{x} \rfloor) \& 1$
if $\pi = 1$ **then**
  $\dot{\boldsymbol{x}} \leftarrow (1,1,1) - \dot{\boldsymbol{x}}$
end if
if $(-1,-1,-1) \cdot \dot{\boldsymbol{x}} > -1$ **then**
  $i \leftarrow 0$
else if $(1,1,-1) \cdot \dot{\boldsymbol{x}} > 1$ **then**
  $i \leftarrow 1$
else if $(1,-1,1) \cdot \dot{\boldsymbol{x}} > 1$ **then**
  $i \leftarrow 2$
else if $(-1,1,1) \cdot \dot{\boldsymbol{x}} > 1$ **then**
  $i \leftarrow 3$
else
  $i \leftarrow 4$
end if
if $i = 4$ **then**
  $\tau \leftarrow 1$
else
  $\tau \leftarrow 0$
end if
if $\pi = 1$ **then**
  $i \leftarrow i + 5$
end if



$\dot{\boldsymbol{x}} \leftarrow \boldsymbol{x} - \lfloor \boldsymbol{x} \rfloor$
$\pi \leftarrow ((1,1,1) \cdot \lfloor \boldsymbol{x} \rfloor) \& 1$
$\dot{\boldsymbol{x}} \leftarrow \dot{\boldsymbol{x}} + \pi((1,1,1) - 2\dot{\boldsymbol{x}})$
$\boldsymbol{v} \leftarrow ((-1,-1,-1) \cdot \dot{\boldsymbol{x}} > -1, (1,1,-1) \cdot \dot{\boldsymbol{x}} > 1,$
$\quad (1,-1,1) \cdot \dot{\boldsymbol{x}} > 1, (-1,1,1) \cdot \dot{\boldsymbol{x}} > 1)$
$\tau \leftarrow 1 - (1,1,1,1) \cdot \boldsymbol{v}$
$i \leftarrow (0,1,2,3) \cdot \boldsymbol{v} + 4\tau + 5\pi$

i: index (0-9)
τ: type (0: red, 1: blue)
π: parity (0: $\lfloor \boldsymbol{x} \rfloor \in \mathbb{Z}_{\mathrm{fcc}}$, 1: $\lfloor \boldsymbol{x} \rfloor \notin \mathbb{Z}_{\mathrm{fcc}}$)

Encoding Transformation

$$T_1(\boldsymbol{x}) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad T_2(\boldsymbol{x}) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$T_3(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad T_5(\boldsymbol{x}) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

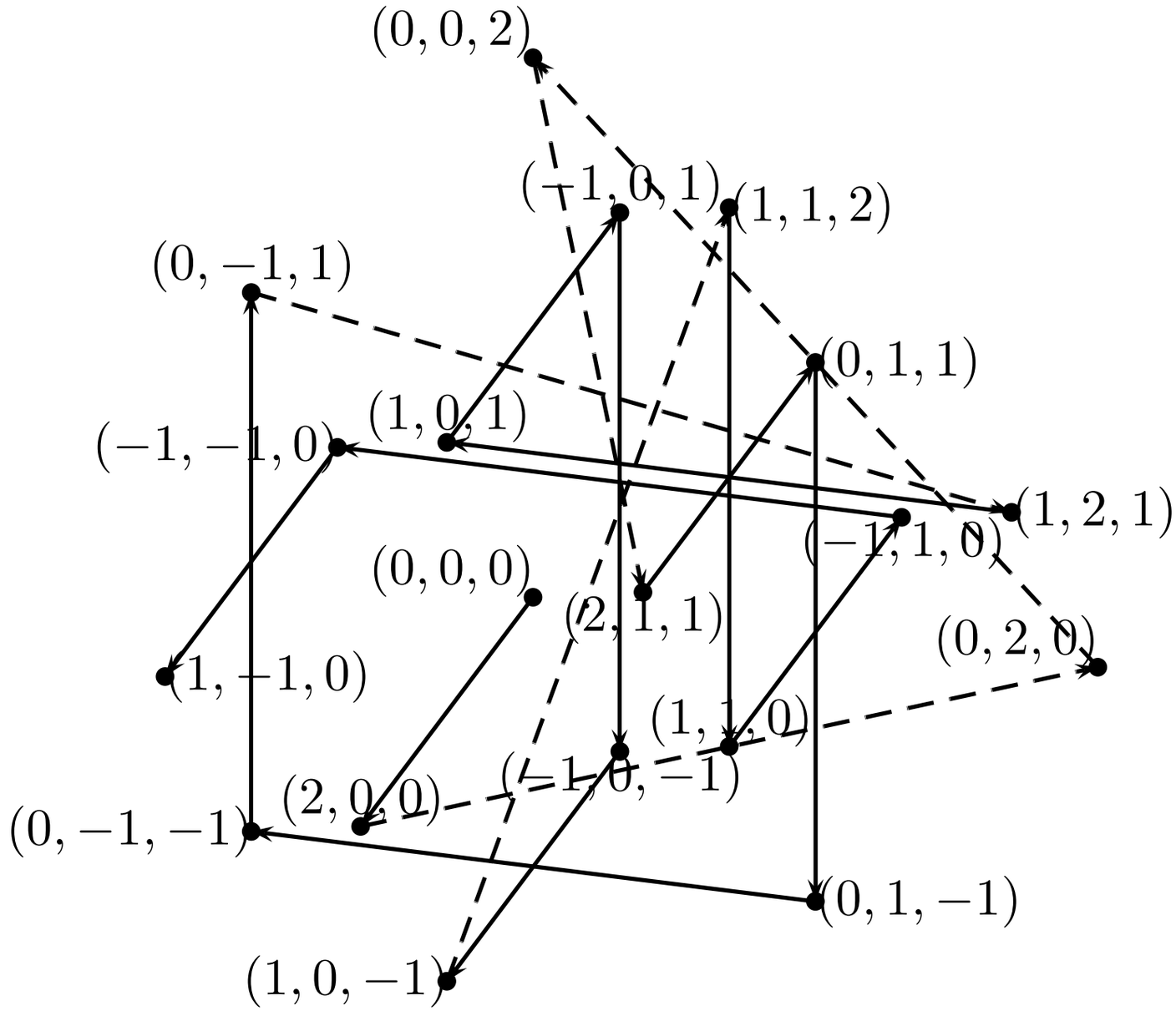
$$T_6(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad T_7(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$T_8(\boldsymbol{x}) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad T_9(\boldsymbol{x}) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$T_i(\boldsymbol{x}) = R\boldsymbol{x} + \boldsymbol{y}$$

$$\begin{aligned} \boldsymbol{y} &\leftarrow ((0,1,1,1) \cdot \boldsymbol{v})((1,1,1) - (\boldsymbol{v}_4, \boldsymbol{v}_3, \boldsymbol{v}_2)) \\ \boldsymbol{y} &\leftarrow \boldsymbol{y} + \pi((1,1,1) - 2\boldsymbol{y}) \\ \boldsymbol{r} &\leftarrow (1,1,1) - 2\boldsymbol{y} \\ R &\leftarrow \begin{bmatrix} \boldsymbol{r}_1 & 0 & 0 \\ 0 & \boldsymbol{r}_2 & 0 \\ 0 & 0 & \boldsymbol{r}_3 \end{bmatrix} \end{aligned}$$

Axis-Aligned Fetching



GLSL Code

```
ivec3 T[10] = ivec3[10](
  ivec3(0,0,0),ivec3(1,1,0),ivec3(1,0,1),
  ivec3(0,1,1),ivec3(0,0,0),ivec3(1,1,1),
  ivec3(0,0,1),ivec3(0,1,0),ivec3(1,0,0),
  ivec3(1,1,1));
ivec3 R[10] = ivec3[10](
  ivec3( 1, 1, 1),ivec3(-1,-1, 1),ivec3(-1, 1,-1),
  ivec3( 1,-1,-1),ivec3( 1, 1, 1),ivec3(-1,-1,-1),
  ivec3( 1, 1,-1),ivec3( 1,-1, 1),ivec3(-1, 1, 1),
  ivec3(-1,-1,-1));
ivec3 stencil[19] = ivec3[19](
    ivec3( 0, 0, 0),ivec3( 2, 0, 0),ivec3( 0, 2, 0),
    ivec3( 0, 0, 2),ivec3( 2, 1, 1),ivec3( 0, 1, 1),
    ivec3( 0, 1,-1),ivec3( 0,-1,-1),ivec3( 0,-1, 1),
    ivec3( 1, 2, 1),ivec3( 1, 0, 1),ivec3(-1, 0, 1),
    ivec3(-1, 0,-1),ivec3( 1, 0,-1),ivec3( 1, 1, 2),
    ivec3( 1, 1, 0),ivec3(-1, 1, 0),ivec3(-1,-1, 0),
    ivec3( 1,-1, 0));
vec3  p_local;
ivec3 origin;
origin = ivec3(floor(p_in));
p_cube = p_in-vec3(origin);
parity = (origin.x+origin.y+origin.z)&1;
p_local = p_cube;
if(parity==1) p_cube = ivec3(1,1,1) - p_cube;
if(dot(p_cube,vec3(-1,-1,-1))>-1)     itet = 0;
else if(dot(p_cube,vec3(1,1,-1))>1)   itet = 1;
else if(dot(p_cube,vec3(1,-1,1))>1)   itet = 2;
else if(dot(p_cube,vec3(-1,1,1))>1)   itet = 3;
else                                  itet = 4;
if(itet==4) type=1;
else        type=0;
vitet = vec4(itet==0,itet==1,itet==2,itet==3);
if(parity==1) itet+=5;
p_local = p_local*R[itet] + T[itet];
origin += T[itet];
ivec3 fcc;
for(int i=0 ; i<19 ; i++)
{
  fcc = origin + R[itet]*stencil[i];
  FETCH(i)
}
```

lookup tables

conditional branches

vector-vector multiplication

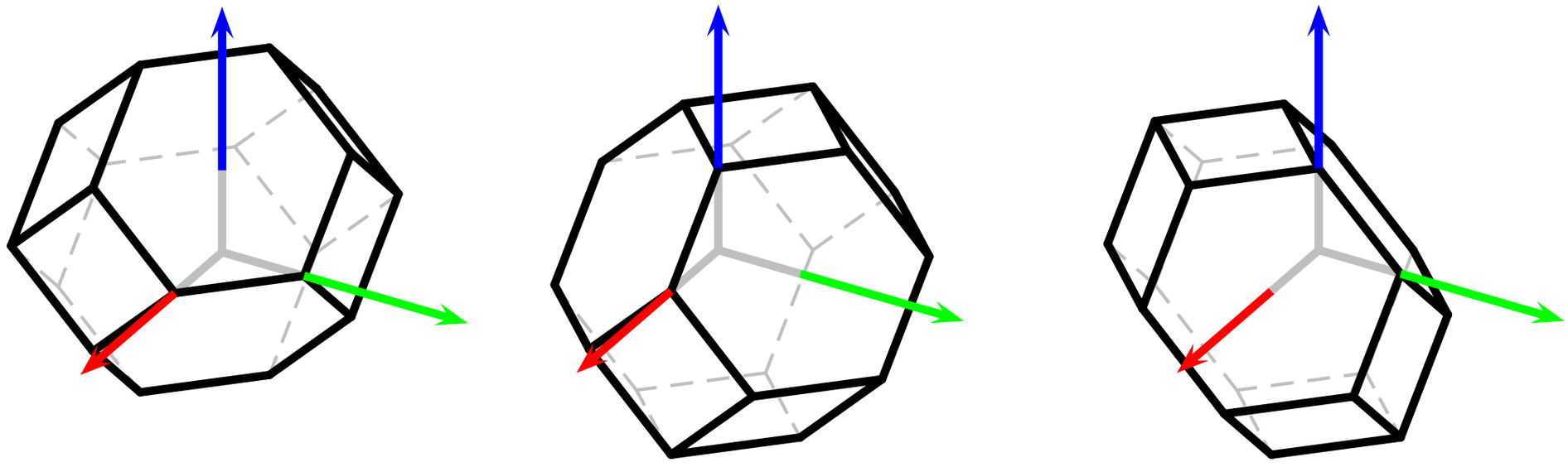
```
vec3  p_local;
ivec3 R;
ivec3 f;
f = ivec3(floor(p_in));
p_cube = p_in-vec3(f);
parity = (f.x+f.y+f.z)&1;
p_local = p_cube;
p_cube += float(parity)*(1-2*p_cube);
vitet = vec4( dot(p_cube,vec3(-1,-1,-1))>-1,
              dot(p_cube,vec3( 1, 1,-1))> 1,
              dot(p_cube,vec3( 1,-1, 1))> 1,
              dot(p_cube,vec3(-1, 1, 1))> 1);
type = 1-(vitet.x+vitet.y+vitet.z+vitet.w);
itet = int(dot(vitet.yzw,vec3(1,2,3)))
     + 4*int(type) + parity*5;
ivec3 offset = int(vitet.y+vitet.z+vitet.w)
             *(1-ivec3(vitet.wzy));
offset += parity*(1 - 2*offset);
R = 1-2*offset;
p_local = vec3(offset) + vec3(R)*p_local;
f += offset;
R *= 2;

                                          FETCH(0)
f.x+=R.x;                                 FETCH(1)
f.x-=R.x; f.y+=R.y;                       FETCH(2)
f.y-=R.y; f.z+=R.z;                       FETCH(3)
f.x+=R.x; f.y+=R.y>>1;  f.z-=R.z>>1;      FETCH(4)
f.x-=R.x;                                 FETCH(5)
f.z-=R.z;                                 FETCH(6)
f.y-=R.y;                                 FETCH(7)
f.z+=R.z;                                 FETCH(8)
f.x+=R.x>>1;  f.y+=(3*R.y)>>1;            FETCH(9)
f.y-=R.y;                                 FETCH(10)
f.x-=R.x;                                 FETCH(11)
f.z-=R.z;                                 FETCH(12)
f.x+=R.x;                                 FETCH(13)
f.y+=R.y>>1;  f.z+=(3*R.z)>>1;            FETCH(14)
f.z-=R.z;                                 FETCH(15)
f.x-=R.x;                                 FETCH(16)
f.y-=R.y;                                 FETCH(17)
f.x+=R.x;                                 FETCH(18)
```

x10-16 performance improvement

Normal Computation

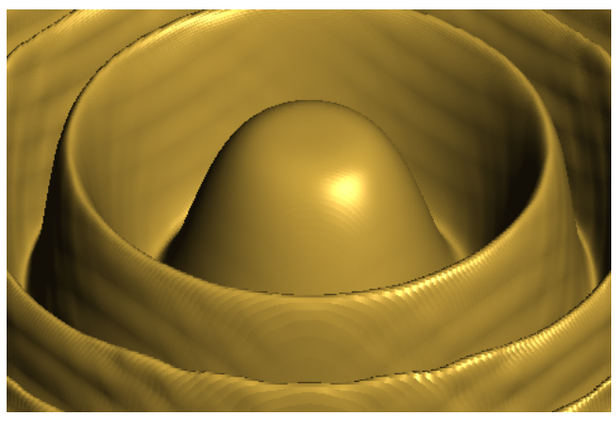
- $D_{\boldsymbol{\xi}} M_{\boldsymbol{\Xi}} = \nabla_{\boldsymbol{\xi}} M_{\boldsymbol{\Xi} \setminus \{\boldsymbol{\xi}\}}$
  - A Directional derivative is a backward difference of another box-spline

- Three additional quadratic box-splines required

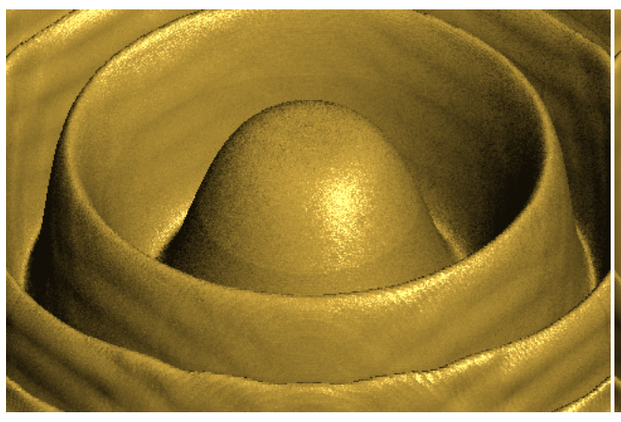


Normal Computation

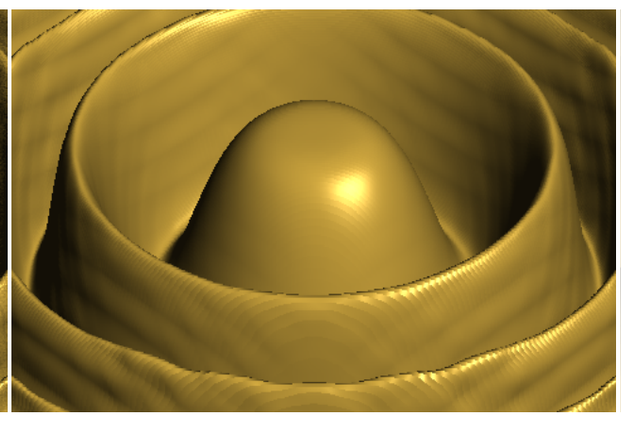
Analytic	Finite difference
No additional data fetch Accurate	19x6=114 additional data fetch Sensitive to δ
Additional code	Evaluation code reused
Fast (up to x1.3)	Slow



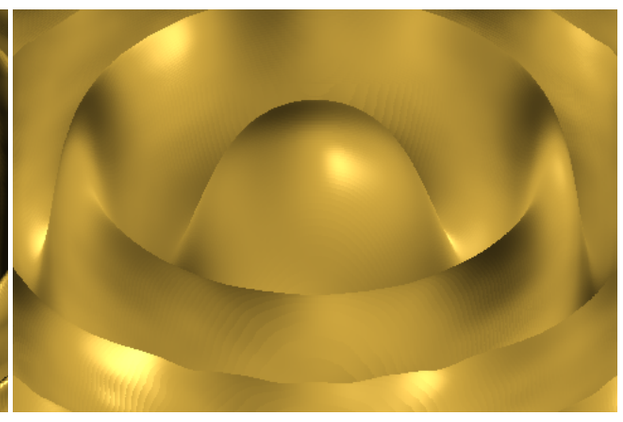
analytic



$\delta=10^{-7}$



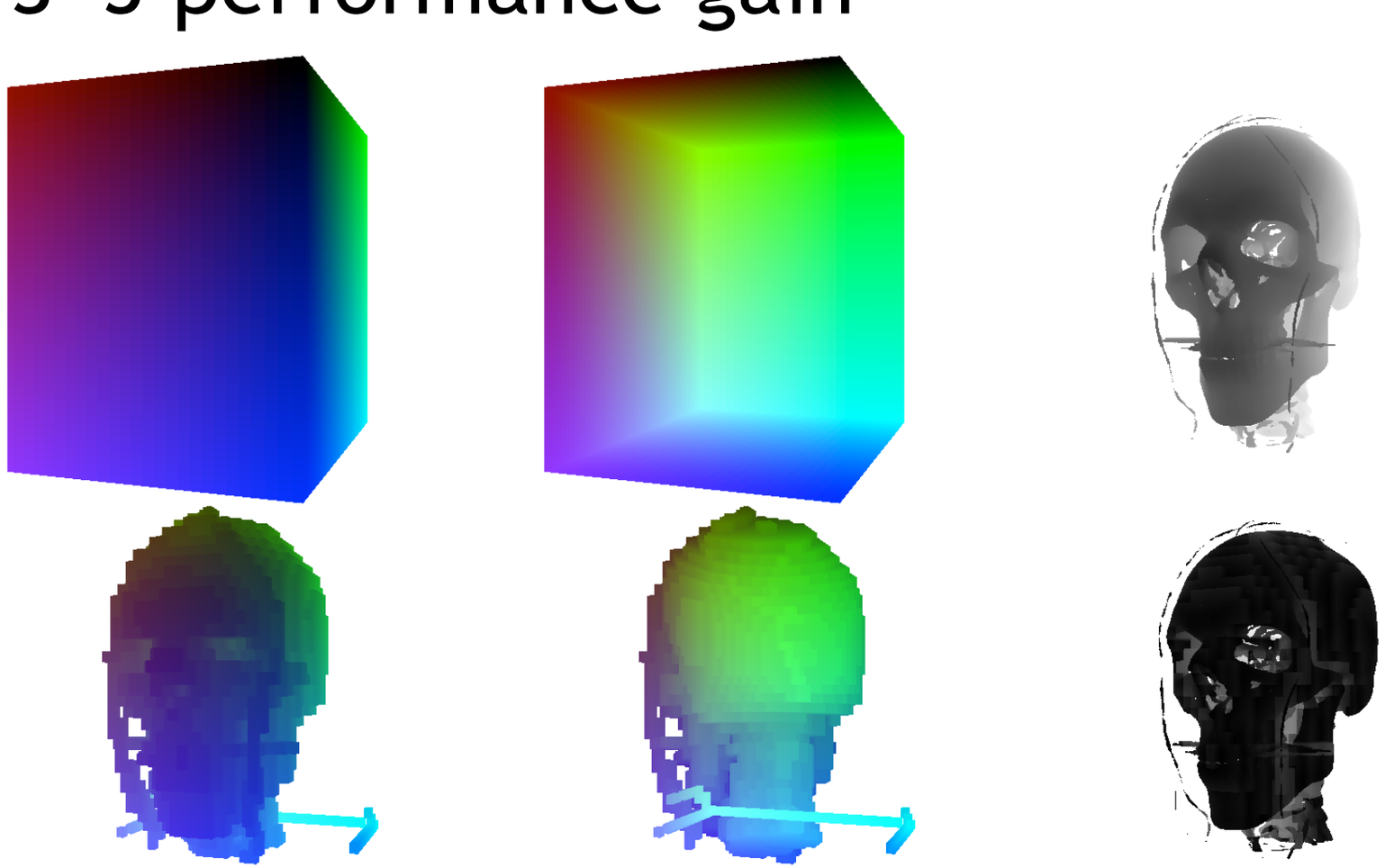
$\delta=10^{-3}$



$\delta=10^{-1}$

Empty-Space Skipping

- Built from BB-coefficients
- Can be done quickly using OpenCL
- x2.3~5 performance gain



Storing FCC Datasets in the GPU Memory

- Split into four Cartesian lattices → not cache-friendly
- Can be stored in a four-channel 3D texture by grouping $\{\boldsymbol{j}, \boldsymbol{j}+(0,1,1), \boldsymbol{j}+(1,0,1), \boldsymbol{j}+(1,1,0)\}$
- No channel selection on current GPU

```
texelFetch(tex,x>>1,y>>1,z>>1)[((x&1)<<1)+(y&1)];
```

- Should be implemented as follows

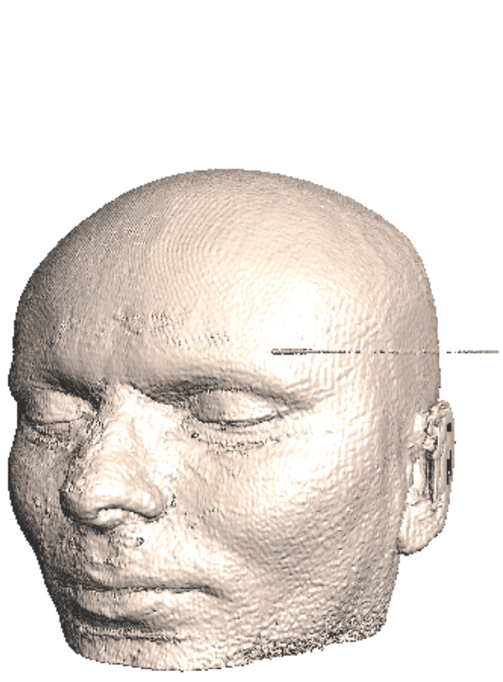
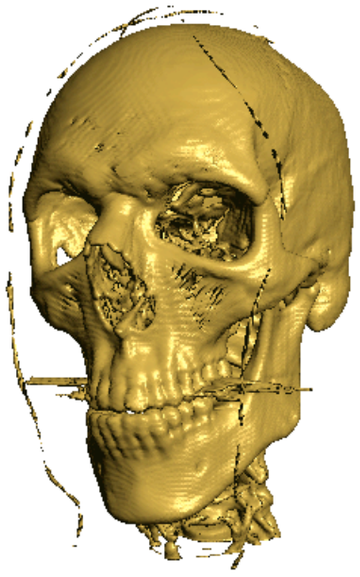
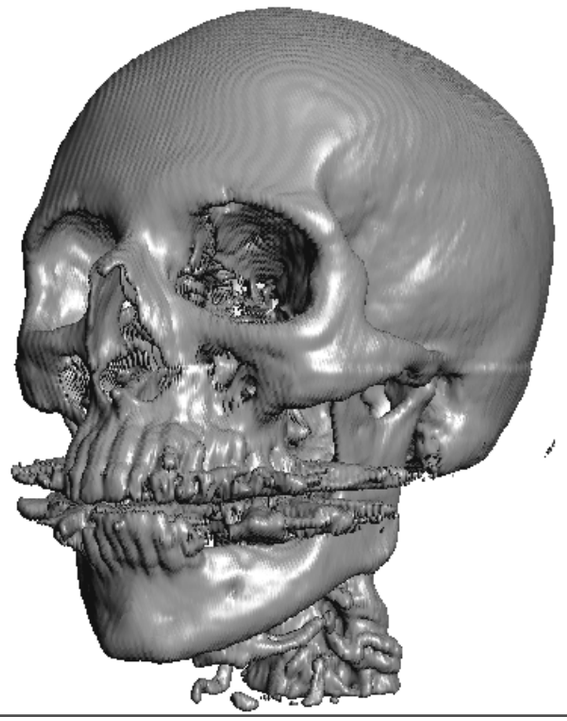
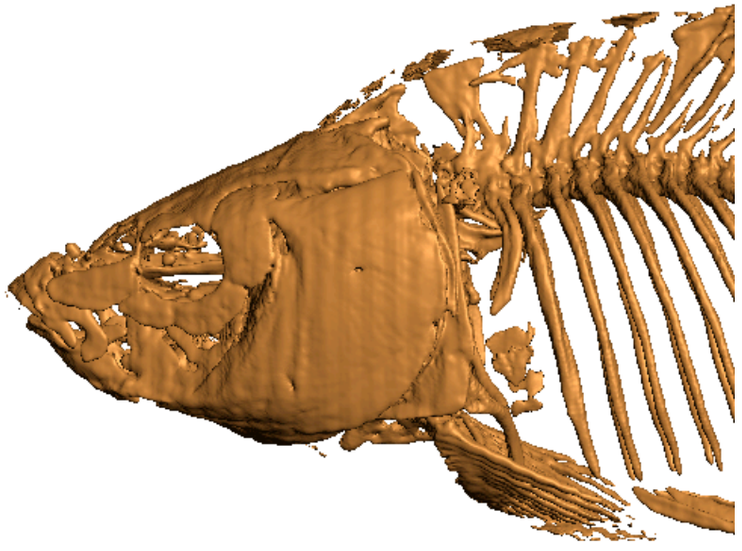
```
i = 2*(x&1) + (y&1);

dot(texelFetch(tex,x>>1,y>>1,z>>1),vec4(i==0,i==1,i==2,i==3));
```

- Slows down the performance

Results

dataset	MRI-Head	VisMale	CT-Head	Carp
size	128x128x128x4	64x128x128x4	128x128x57x4	128x128x256x4
fps	65.2	68.9	54.6	22.3

- Intel® Core™ i7 860 @2.80 GHz, Windows 7 Professional (64 bit), 8GB memory, NVIDIA GeForce GTX 460 (driver 285.86).
- Datasets courtesy of “the vollib library”

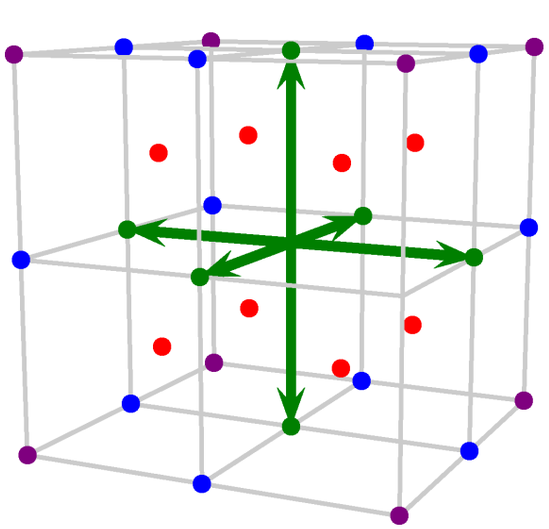





Results

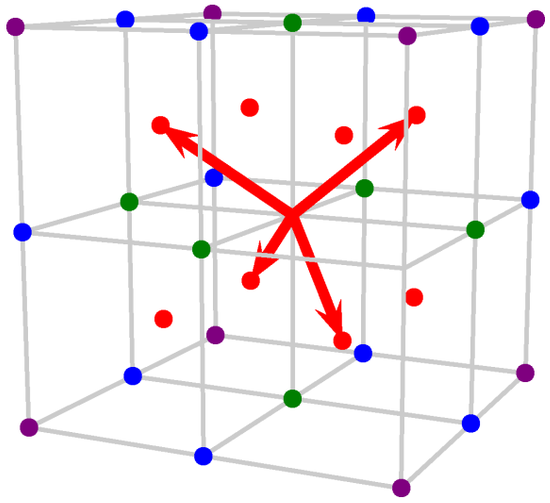
method	lattice	size	fps
[Kim 2012]	Cartesian	50x50x101x4=1,010,000	25.11
[Finkbeiner et al. 2010]	BCC	64x64x128x2=1,048,576	10.66
[Sigg & Hadwiger 2005]	FCC	80x80x160=1,024,000	61.14

BCC Datasets

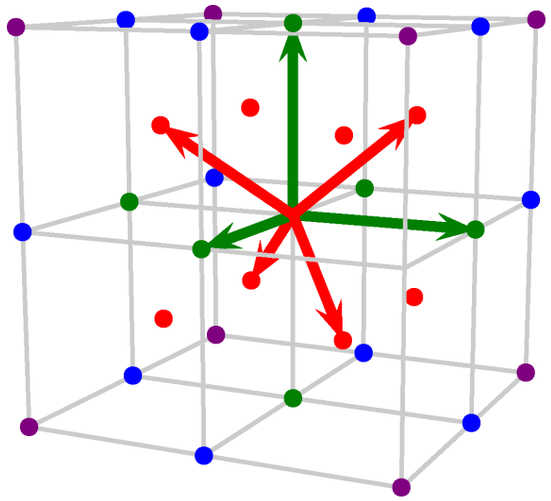
Quartic Box-Spline on the BCC Lattice



bcc12
[Csébfalvi and Hadwiger 2006]

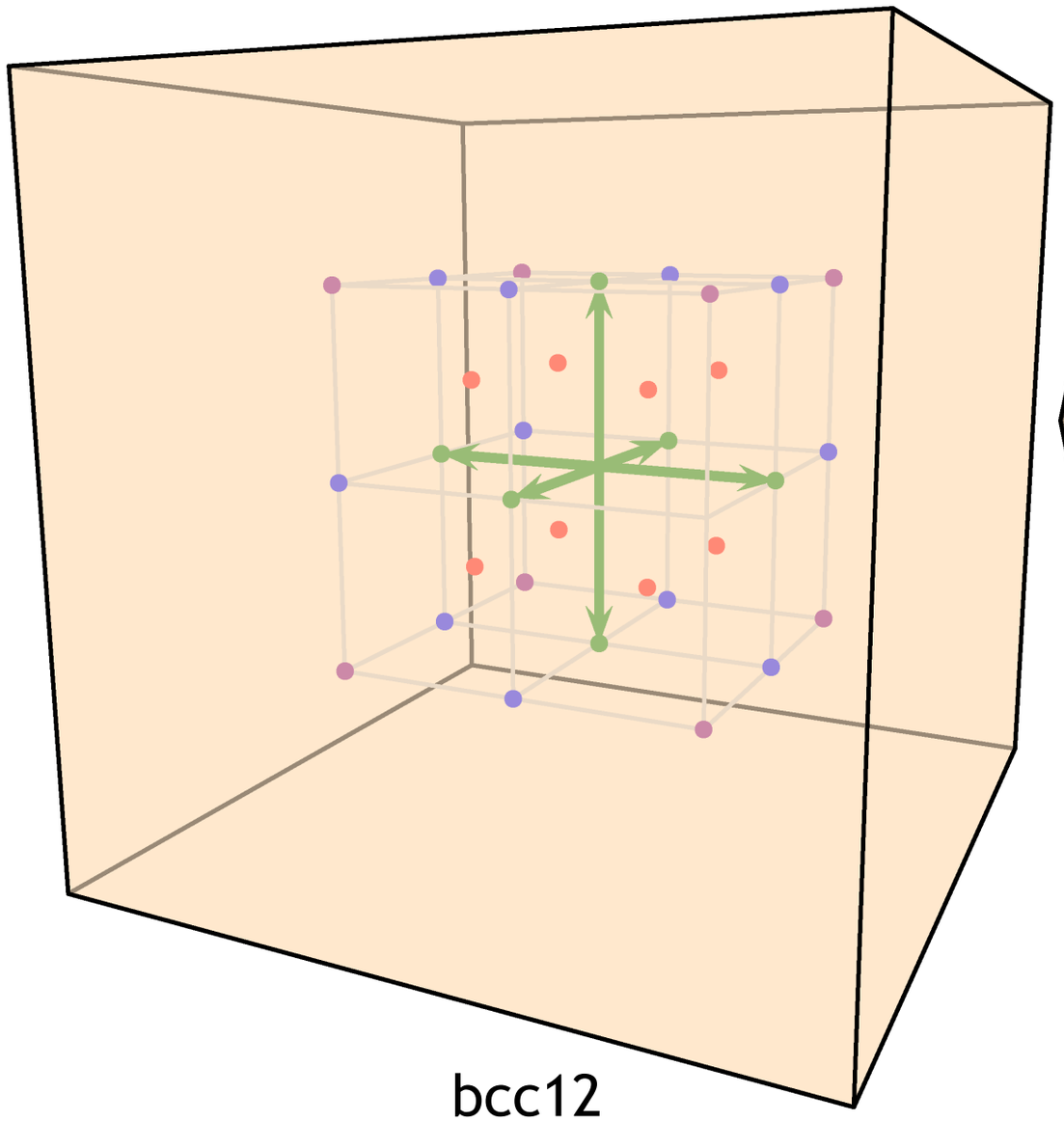


bcc8
[Entezari et al. 2004]

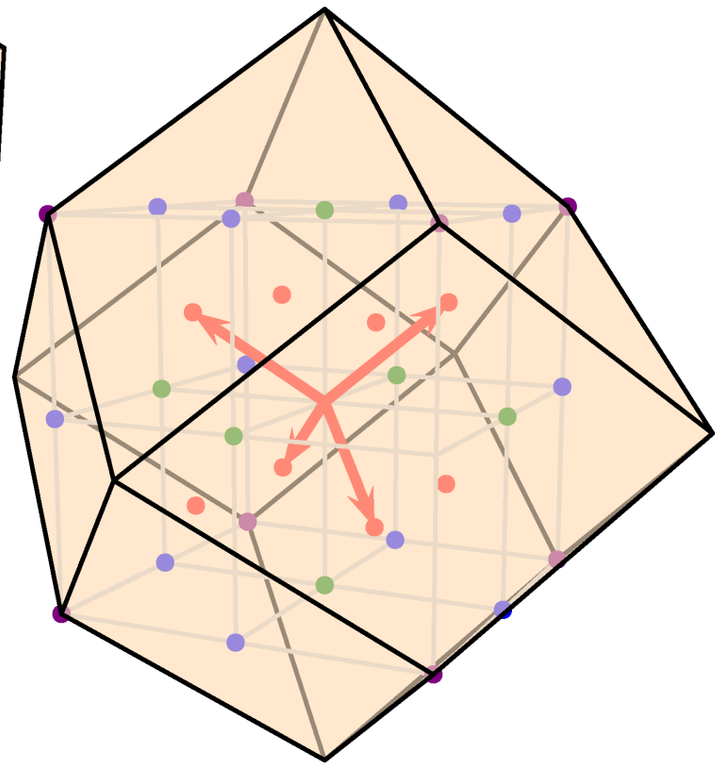


bcc7
[Kim 2013]

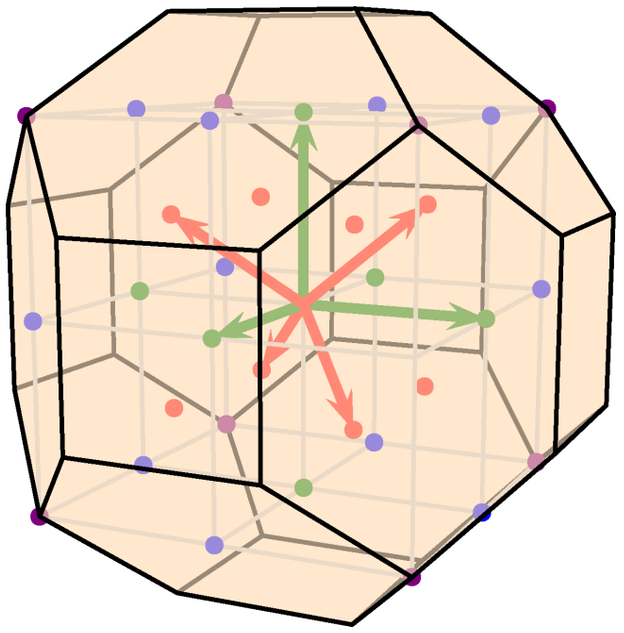
Quartic Box-Spline on the BCC Lattice



bcc12
[Csébfalvi and Hadwiger 2006]



bcc8
[Entezari et al. 2004]

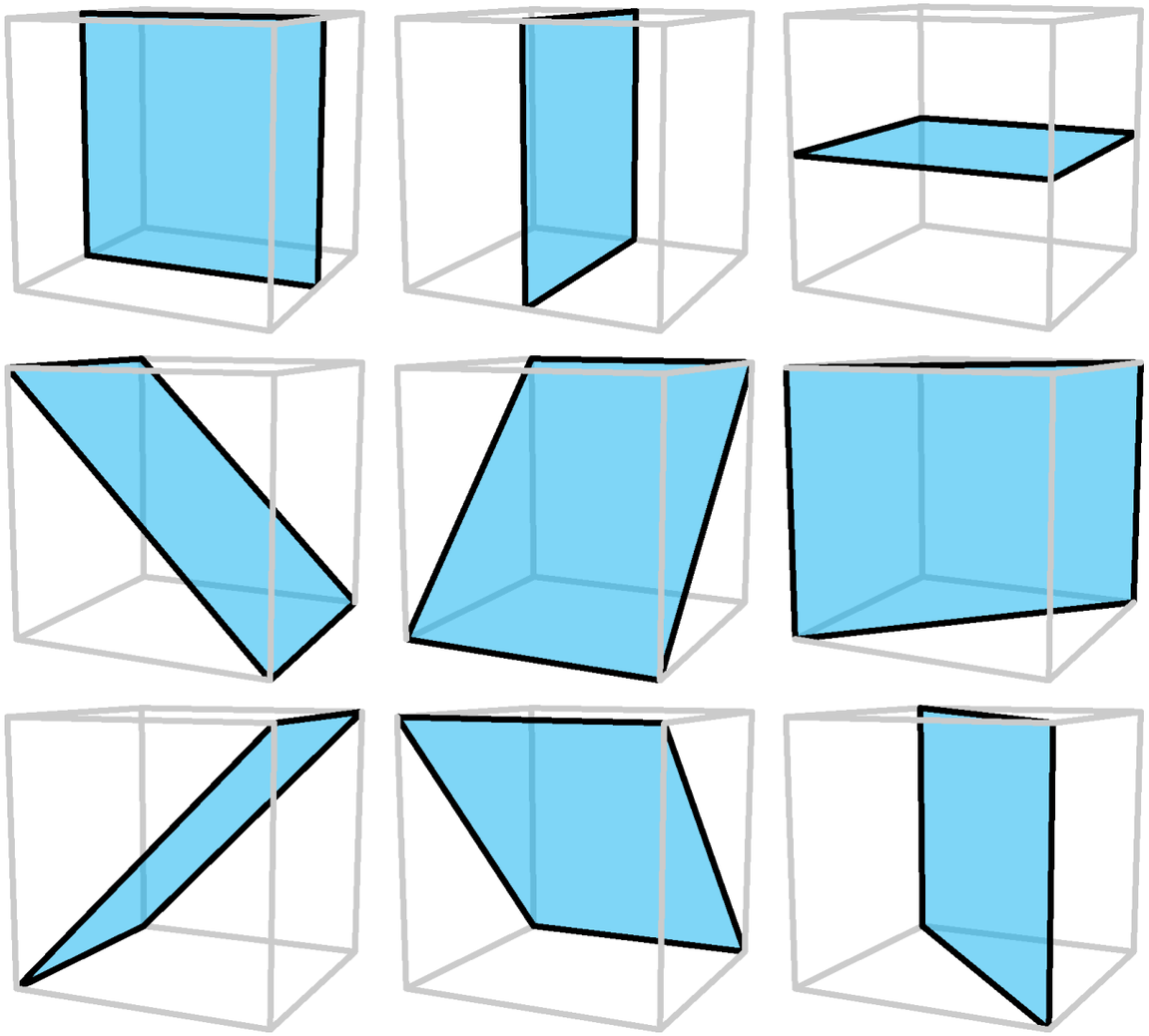


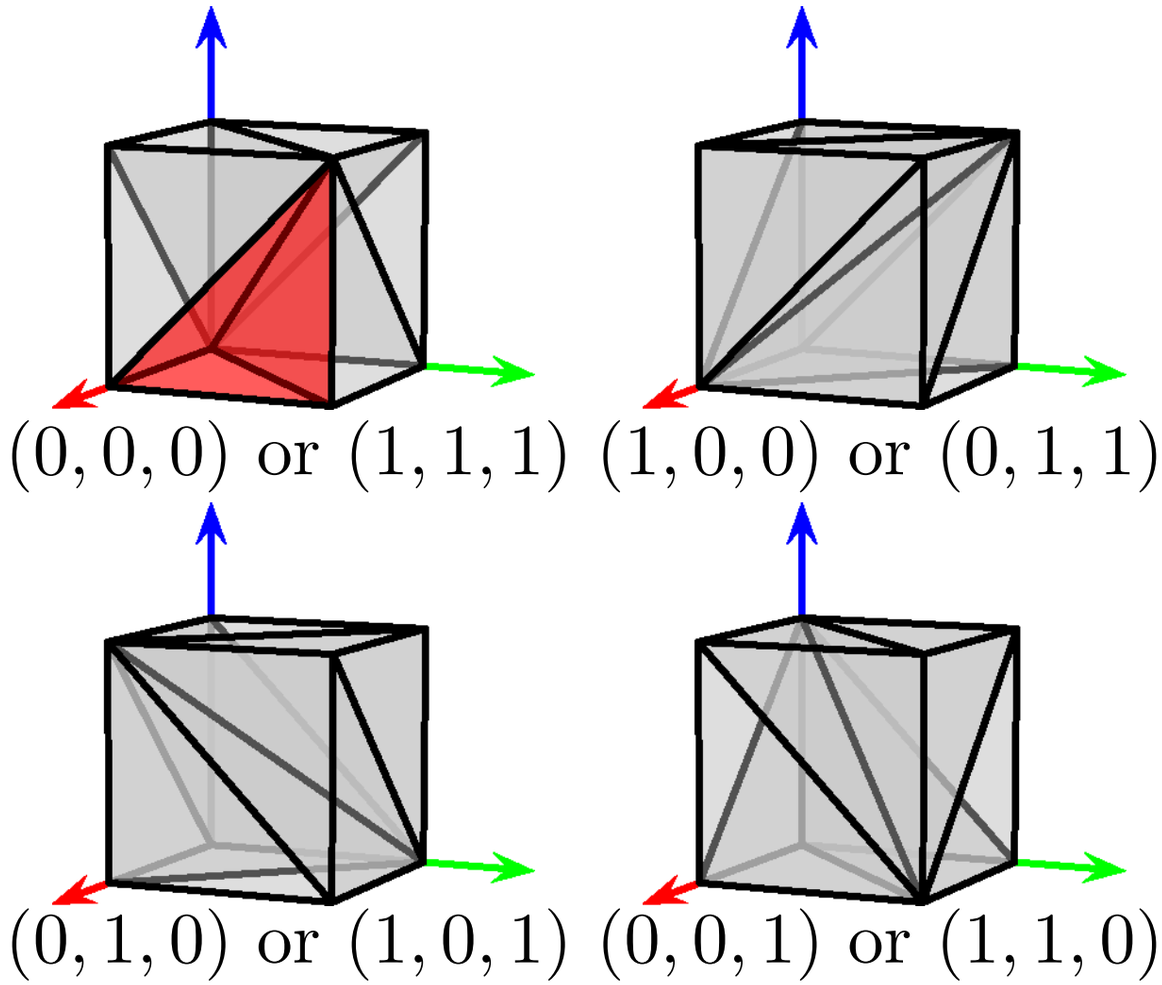
bcc7
[Kim 2013]

Comparison

	bcc12	bcc8	bcc7
degree	9	5	4
approximation order	4	4	4
shift-invariant poly pieces	1	6	24
volume of support	512	128	120
stencil size	128	32	30
Reisz basis?	no	yes	no

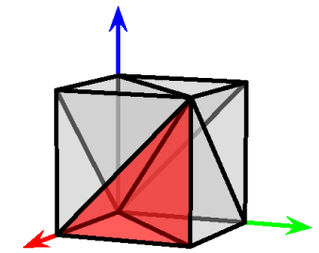
Polynomial Structure



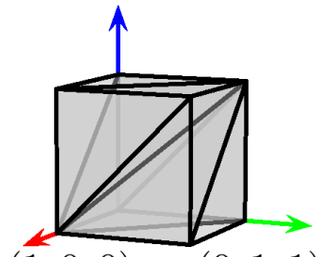


$$\boldsymbol{\pi} := \lfloor \boldsymbol{x} \rfloor \bmod 2$$

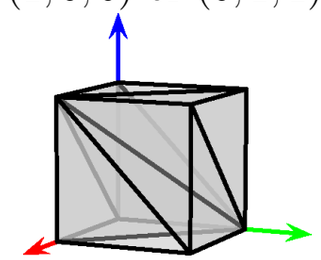
Transformation



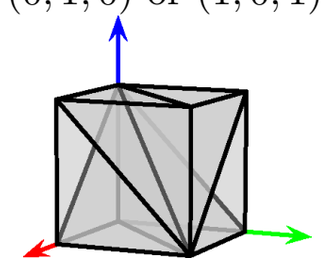
$(0,0,0)$ or $(1,1,1)$



$(1,0,0)$ or $(0,1,1)$

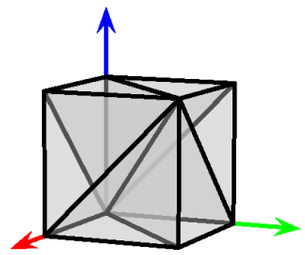


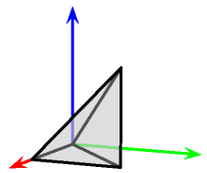
$(0,1,0)$ or $(1,0,1)$



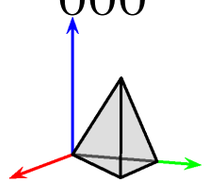
$(0,0,1)$ or $(1,1,0)$

$R_{\boldsymbol{\pi}}(\boldsymbol{x}-\boldsymbol{\pi})$

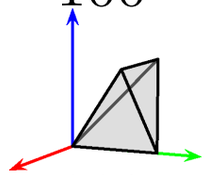




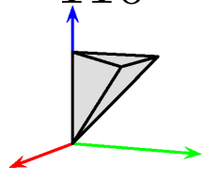
000



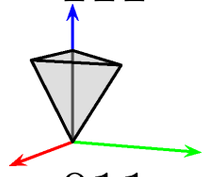
100



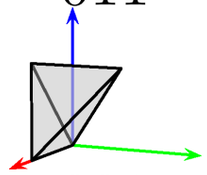
110



111

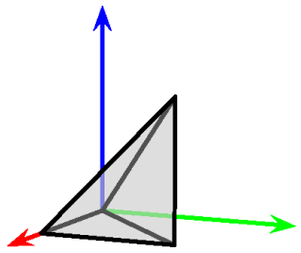


011



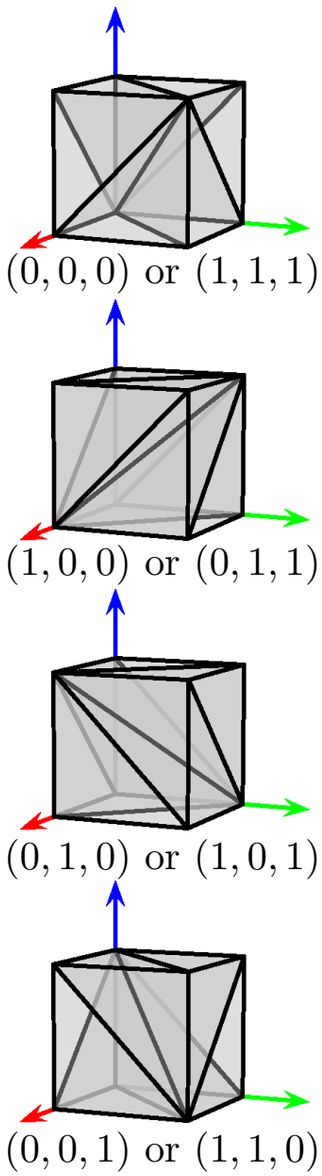
001

P_{τ}



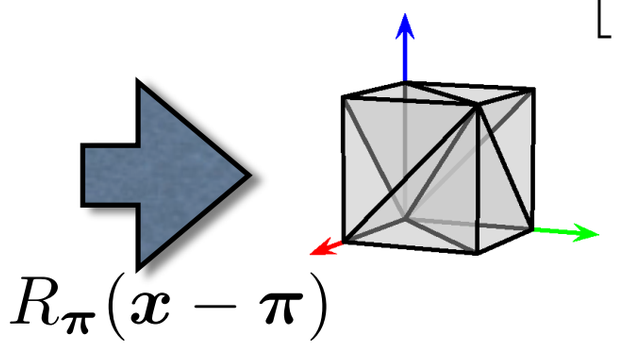
Translation ➜ Reflection ➜ Permutation

Encoding Translation & Reflection



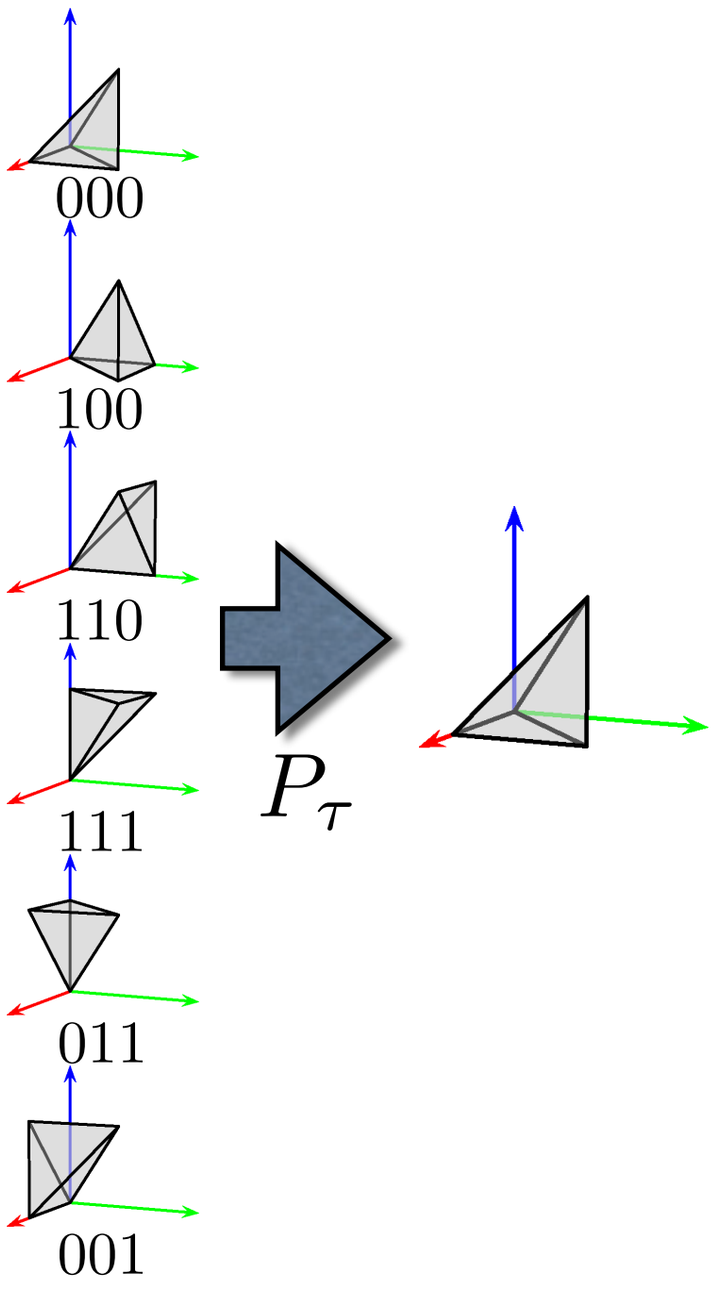
$$R_{000} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_{001} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad R_{010} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_{100} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{111} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad R_{110} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_{101} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad R_{011} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$



$$R_{\boldsymbol{\pi}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - 2 \begin{bmatrix} \boldsymbol{\pi}_1 & 0 & 0 \\ 0 & \boldsymbol{\pi}_2 & 0 \\ 0 & 0 & \boldsymbol{\pi}_3 \end{bmatrix}$$

Encoding Permutation

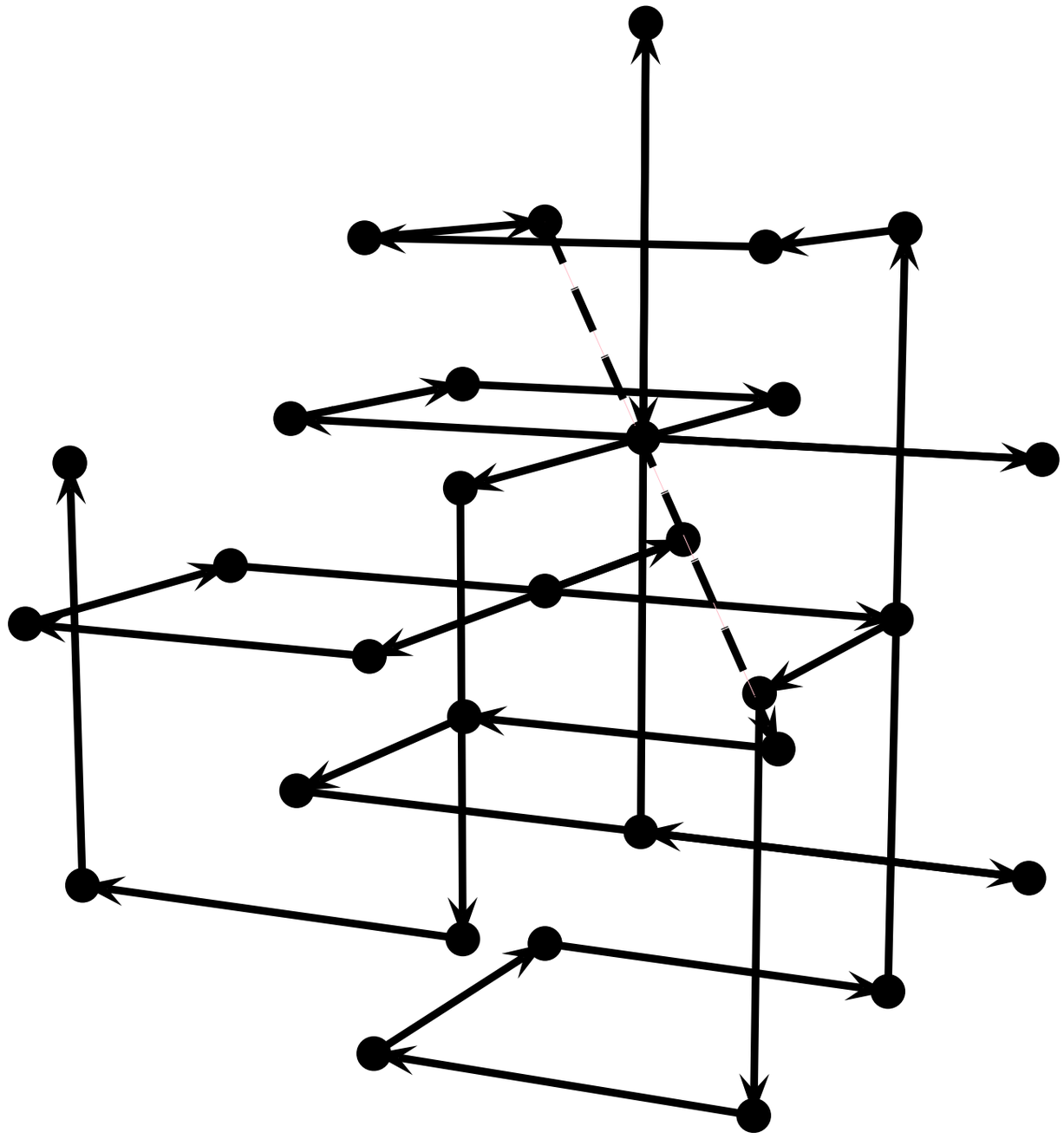


$$P_{000} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad P_{100} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad P_{110} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$P_{111} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad P_{011} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad P_{001} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$P_{\boldsymbol{\tau}} = \begin{bmatrix} !(\boldsymbol{\tau}_1 | \boldsymbol{\tau}_2) & \boldsymbol{\tau}_1 \& !\boldsymbol{\tau}_3 & \boldsymbol{\tau}_2 \& \boldsymbol{\tau}_3 \\ ? & ? & ? \\ \boldsymbol{\tau}_1 \& \boldsymbol{\tau}_2 & !\boldsymbol{\tau}_1 \& \boldsymbol{\tau}_3 & !(\boldsymbol{\tau}_2 | \boldsymbol{\tau}_3) \end{bmatrix}$$

Axis-Aligned Fetching



GLSL Code

```
#define FETCH(var)  var = texelFetch(volume_tex, o, 0).r;
#define IT0 itv1.x
#define IT1 itv1.y
#define IT3 itv1.z
#define IT4 itv2.x
#define IT6 itv2.y
#define IT7 itv2.z
  float c[30];
  ivec3 o;
  vec3  xtet;
  ivec3 xlow = ivec3(p_in);
  ivec3 parity = ivec3(xlow.x&0x01, xlow.y&0x01, xlow.z&0x01);
  o = ivec3(xlow) + parity;
  ivec3 Rc = ivec3(1,1,1)-2*parity;
  vec3  xc = Rc*(p_in-o);
  int itet = (int(xc.y >= xc.x)<<2) + (int(xc.z >= xc.x)<<1) + int(xc.z >= xc.y);
  ivec3 itv1 = ivec3(itet==0, itet==1, itet==3);
  ivec3 itv2 = ivec3(itet==4, itet==6, itet==7);
  xtet = IT0*xc.xyz + IT1*xc.xzy + IT3*xc.zxy + IT4*xc.yxz + IT6*xc.yzx + IT7*xc.zyx;
  ivec3 R[3] = ivec3[](
             2*ivec3(Rc.x*(IT0+IT1), Rc.y*(IT4+IT6), Rc.z*(IT3+IT7)),
             2*ivec3(Rc.x*(IT3+IT4), Rc.y*(IT0+IT7), Rc.z*(IT1+IT6)),
             2*ivec3(Rc.x*(IT6+IT7), Rc.y*(IT1+IT3), Rc.z*(IT0+IT4)));
           FETCH(c01); o -= R[0];     FETCH(c11); o += (R[0]<<1); FETCH(c10);
  o -= R[1];     FETCH(c21); o -= R[0];     FETCH(c13); o += (R[1]<<1); FETCH(c12);
  o += R[0];     FETCH(c20); o -= R[2];     FETCH(c30); o -= R[1];     FETCH(c19);
  o -= R[0];     FETCH(c15); o += R[1];     FETCH(c17); o += (R[2]<<1); FETCH(c16);
  o += R[0];     FETCH(c29); o -= R[1];     FETCH(c18); o -= R[0];     FETCH(c14);
  o += ((-R[0]+R[1]-3*R[2])>>1);  FETCH(c07);
  o -= R[1];     FETCH(c09); o += R[0];     FETCH(c05); o += (R[1]<<1); FETCH(c27);
  o -= R[1];     FETCH(c03); o += (R[2]<<1); FETCH(c28); o -= R[2];     FETCH(c02);
  o += R[1];     FETCH(c26); o -= (R[1]<<1); FETCH(c04); o -= R[0];     FETCH(c08);
  o += R[1];     FETCH(c06); o += (R[0]<<1); FETCH(c22); o -= R[2];     FETCH(c23);
  o -= R[1];     FETCH(c25); o += R[2];     FETCH(c24);
```

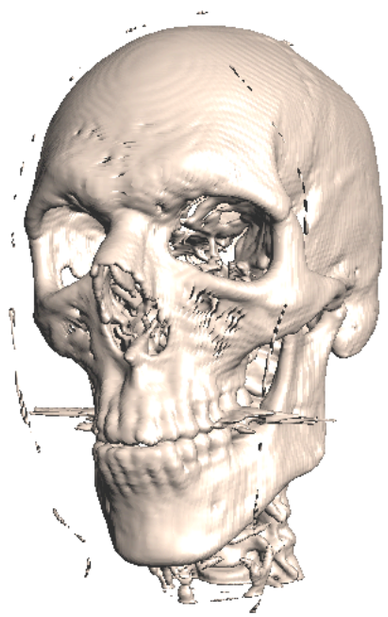
Simplified Symbolic Formula

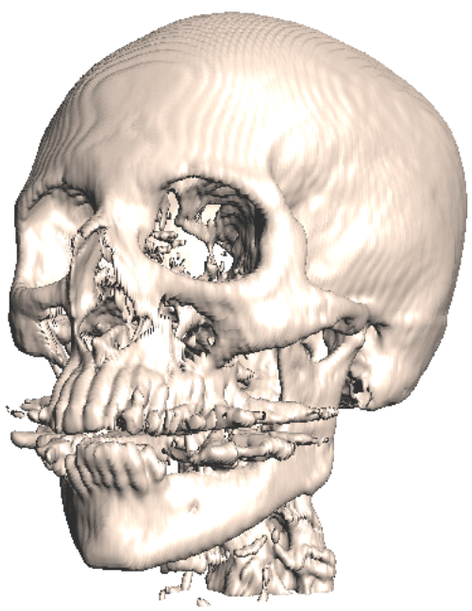
$$
\begin{aligned}
&8(\\
&\quad -x^4(c_{10}+c_{20})\\
&\quad +(-1+x)^4(c_1-c_6-c_7-c_8-c_9+c_{11}+c_{12}+c_{13}+c_{15})\\
&\quad +(3-x)^4c_1\\
&\quad -(-2+x)^4(c_1-c_6+c_{12})\\
&\quad +(1+x)^4c_{10}\\
&\quad +z^4(c_2+c_4-c_{14}-c_{16}-c_{18}+c_{22}+c_{26}+c_{28}-c_{29})\\
&\quad -(-1+z)^4(c_3+c_5)\\
&\quad -(1+z)^4(c_2+c_4-c_{14})\\
&\quad +(-2+z)^4c_3\\
&\quad +(2+z)^4c_2\\
&)\\
&+4(\\
&\quad -(x+y)^3(2+x-y)(c_{10}-c_{20})\\
&\quad -(-2+x-y)^3(-4+x+y)(c_1-c_{12})\\
&\quad +(-2+y+z)^3(2+y-z)(c_3-c_5)\\
&\quad +(-2+y-z)^3(2+y+z)(c_2-c_4)\\
&)\\
&+2(\\
&\quad -(x-y)^3\left(4\left(c_{10}-c_{21}\right)-2z\left(c_{24}-c_{25}\right)-(x+y)\left(2\left(c_{20}-c_{21}\right)+c_{24}+c_{25}\right)\right)\\
&\quad -(x+z)^3\left(4\left(c_2+c_4+c_{10}+c_{14}-c_{18}\right)+2y\left(c_2-c_4-c_{10}+c_{20}-c_{22}\right)-(x-z)\left(2\left(c_{14}-c_{18}\right)+c_2+c_4-c_{10}-c_{20}+c_{22}\right)\right)\\
&\quad +(x-z)^3\left(-4\left(c_3+c_5+c_{10}-c_{19}\right)-2y\left(c_3-c_5-c_{10}+c_{20}-c_{23}\right)+(x+z)\left(c_3+c_5-c_{10}-2c_{19}-c_{20}+c_{23}\right)\right)\\
&\quad -(y+z)^3\left(4\left(c_2-c_4+c_{14}-c_{16}\right)-2x\left(c_{14}-c_{16}-c_{18}-c_{22}+c_{29}\right)+(y-z)\left(2\left(c_4-c_{26}\right)-c_{14}+c_{16}-c_{18}-c_{22}+c_{29}\right)\right)\\
&\quad -(y-z)^3\left(4\left(c_3-c_5-c_{17}\right)+2x\left(c_{17}+c_{19}+c_{23}-c_{30}\right)-(y+z)\left(-2\left(c_5-c_{27}\right)+c_{23}-c_{17}+c_{19}-c_{30}\right)\right)\\
&\quad +(-2+x+y)^3\left(4\left(c_1-c_{12}\right)+2\left(c_6+c_7-c_8-c_9+c_{15}\right)+2z\left(c_6-c_7-c_8+c_9-c_{15}\right)-(x-y)\left(c_6+c_7-c_8-c_9+c_{15}-2\left(c_{12}-c_{13}\right)\right)\right)\\
&\quad +(-2+x+z)^3\left(6c_1+2\left(c_3+c_5+c_{12}-c_{15}\right)+4\left(c_6-c_7\right)-2y\left(c_1-c_3+c_5-c_{12}-c_{15}\right)-(x-z)\left(c_1-c_3-c_5+c_{12}+c_{15}+2\left(c_6-c_7\right)\right)\right)\\
&\quad +(-2+x-z)^3\left(6c_1+2\left(c_2+c_4+c_{12}\right)-2y\left(c_1-c_2+c_4-c_{12}\right)-(x+z)\left(c_1-c_2-c_4+c_{12}\right)\right)\\
&)
\end{aligned}
$$

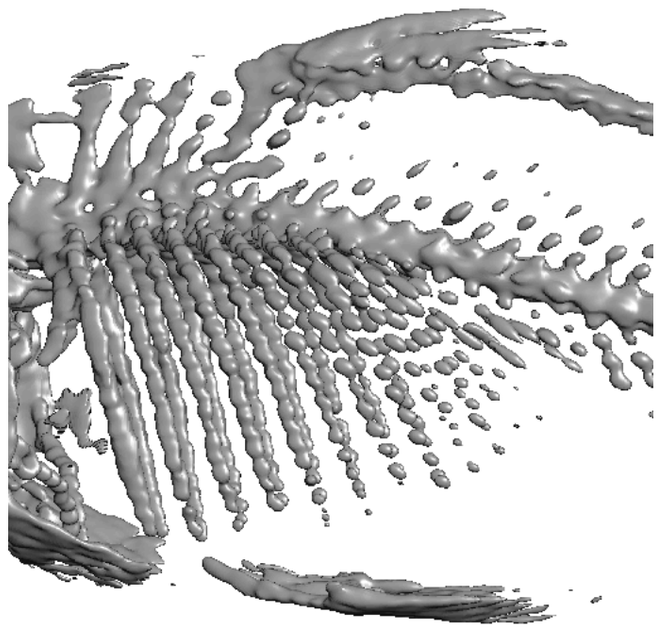
Results

Dataset	Size	Image size	bcc12	bcc8	bcc7	bcc7/bcc8	bcc7/bcc12
`CT-Head`	$128 \times 128 \times 56 \times 2$	512×512	26.5358	10.4807	12.7021	1.2120	0.4787
`Carp` (part)	$64 \times 64 \times 128 \times 2$	512×512	16.0361	6.7642	8.2587	1.2210	0.5150
`MRI-Head`	$128 \times 128 \times 128 \times 2$	512×512	35.2842	13.6075	16.4923	1.2120	0.4674
`VisMale`	$64 \times 128 \times 128 \times 2$	512×512	36.5782	13.5530	16.9610	1.2515	0.4637
ML	$39 \times 39 \times 39 \times 2$	480×320	33.7487	13.7503	16.8196	1.2232	0.4984
ML	$31 \times 31 \times 31 \times 2$	480×320	34.2323	13.8090	16.7790	1.2151	0.4902
ML	$26 \times 26 \times 26 \times 2$	480×320	34.2470	13.7327	16.7058	1.2165	0.4878
ML	$22 \times 22 \times 22 \times 2$	480×320	34.8418	13.9543	16.9900	1.2175	0.4876
ML	$19 \times 19 \times 19 \times 2$	480×320	35.5754	14.2897	17.2601	1.2079	0.4852

- Intel® Core™ i7 860 @2.80 GHz, Windows 7 Professional (64 bit), 8GB memory, NVIDIA GeForce GTX 460 .
- Datasets courtesy of “the vollib library”







Wrap-Up

Conclusion

- To boost the performance, we need a smart workaround to ditch conditional branches & lookup tables.
- Analytic normal computation is efficient & fast.
- Empty space skipping can be easily done and boosts the performance significantly. (Isosurface rendering only)

More Work

- Hierarchical spline on the FCC/BCC lattices
- Surface implicitization using the FCC/BCC lattices & signed distance fields
- Isosurface extraction
- FEM

Thank you