Adaptive Mesh Refinement for Self-Force Calculations

Jonathan Thornburg

in collaboration with

Leor Barack, Norichika Sago, and Darren Golbourn

General Relativity Group
School of Mathematics
University of Southampton

Self-Force Calculations

As discussed by Leor Barack & Norichika Sago in their talks, we (the Southampton group) do self-force calculations by finding the **metric perturbation** in the **Lorenz gauge**:
⟦details in Barack & Sago, *Phys. Rev. D* **75**, 064021 (2007)⟧

Self-Force Calculations

As discussed by Leor Barack & Norichika Sago in their talks, we (the Southampton group) do self-force calculations by finding the **metric perturbation** in the **Lorenz gauge**:
⟦details in Barack & Sago, *Phys. Rev. D* **75**, 064021 (2007)⟧

- decompose the metric perturbation into (ℓ, m) multipole modes
- for each (ℓ, m) multipole mode, **numerically integrate the metric-perturbation equations in the time domain**
- compute the physical self-force from the metric-perturbation multipoles via a regularized mode sum

Self-Force Calculations

As discussed by Leor Barack & Norichika Sago in their talks, we (the Southampton group) do self-force calculations by finding the **metric perturbation** in the **Lorenz gauge**:
⟦details in Barack & Sago, *Phys. Rev. D* **75**, 064021 (2007)⟧

- decompose the metric perturbation into (ℓ, m) multipole modes
- for each (ℓ, m) multipole mode, **numerically integrate the metric-perturbation equations in the time domain**
- compute the physical self-force from the metric-perturbation multipoles via a regularized mode sum

[*Most of the material in this talk would be equally applicable to other self-force calculation schemes, or indeed to other problems involving the numerical solution of time-evolution PDEs.*]

Why Mesh Refinement

After the (ℓ, m) multipole decomposition, our typical metric perturbation equation (to be numerically integrated) looks like

$$\Box\phi + V_\ell(r)\phi = S_{\ell m}(t)\,\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big) \qquad \Rightarrow$$

where $V_\ell(r)$, $S_{\ell m}(t)$, and $\vec{\mathsf{x}}_{\text{particle}}(t)$ are known, and ϕ is a complex field on a Schwarzschild or Kerr background (1 or 2 space dimensions $\times$ time)

Why Mesh Refinement

After the (ℓ, m) multipole decomposition, our typical metric perturbation equation (to be numerically integrated) looks like

$$\Box\phi + V_\ell(r)\phi = S_{\ell m}(t)\,\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big) \qquad \Rightarrow$$

where $V_\ell(r)$, $S_{\ell m}(t)$, and $\vec{\mathsf{x}}_{\text{particle}}(t)$ are known, and ϕ is a complex field on a Schwarzschild or Kerr background (1 or 2 space dimensions $\times$ time)

We need to solve many of these equations (one for each (ℓ, m)) to very high accuracy, as efficiently as possible.

Why Mesh Refinement

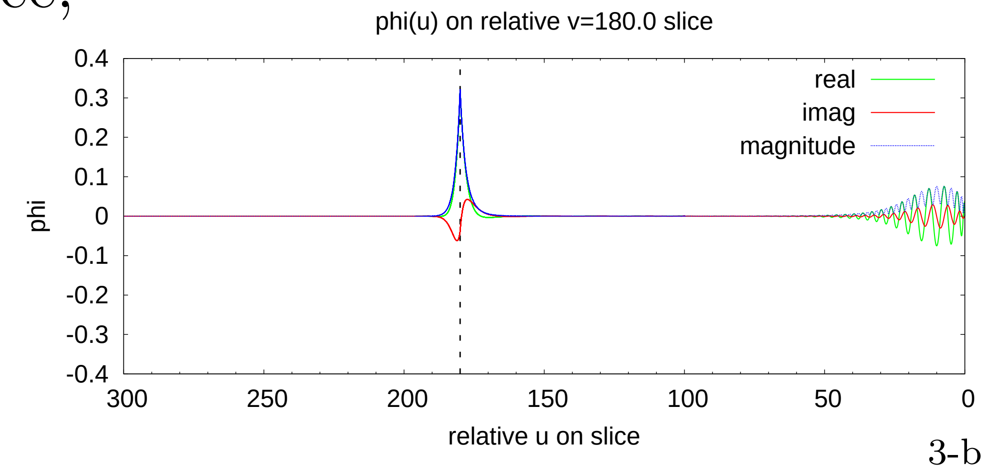
After the (ℓ, m) multipole decomposition, our typical metric perturbation equation (to be numerically integrated) looks like

$$\Box\phi + V_\ell(r)\phi = S_{\ell m}(t)\,\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big) \qquad \Rightarrow$$

where $V_\ell(r)$, $S_{\ell m}(t)$, and $\vec{\mathsf{x}}_{\text{particle}}(t)$ are known, and ϕ is a complex field on a Schwarzschild or Kerr background (1 or 2 space dimensions $\times$ time)

We need to solve many of these equations (one for each (ℓ, m)) to very high accuracy, as efficiently as possible.

Because of the $\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big)$ source, ϕ varies **rapidly** near the particle, but relatively **slowly** elsewhere



Why Mesh Refinement

After the (ℓ, m) multipole decomposition, our typical metric perturbation equation (to be numerically integrated) looks like

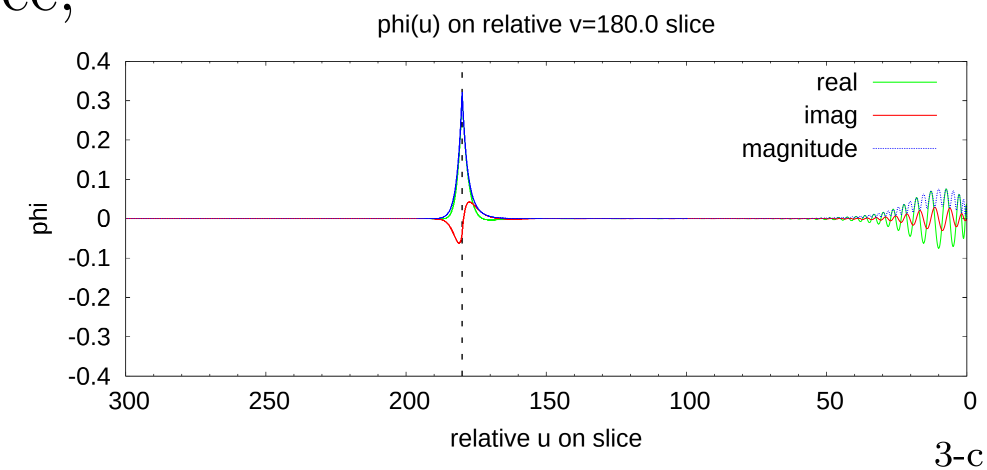
$$\Box\phi + V_\ell(r)\phi = S_{\ell m}(t)\,\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big) \qquad \Rightarrow$$

where $V_\ell(r)$, $S_{\ell m}(t)$, and $\vec{\mathsf{x}}_{\text{particle}}(t)$ are known, and ϕ is a complex field on a Schwarzschild or Kerr background (1 or 2 space dimensions $\times$ time)

We need to solve many of these equations (one for each (ℓ, m)) to very high accuracy, as efficiently as possible.

Because of the $\delta\big(\vec{\mathsf{x}} - \vec{\mathsf{x}}_{\text{particle}}(t)\big)$ source, ϕ varies **rapidly** near the particle, but relatively **slowly** elsewhere

$\Rightarrow$ Want mesh refinement!



Typography of (Finite-Difference) Mesh Refinement

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing $\Rightarrow$ more complicated finite differencing
- rapidly changing resolution $\Rightarrow$ local shortage/surplus of grid points

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing $\Rightarrow$ more complicated finite differencing
- rapidly changing resolution $\Rightarrow$ local shortage/surplus of grid points
- **doesn't generalize to > 1 spatial dimension (grid "tangles")**

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing $\Rightarrow$ more complicated finite differencing
- rapidly changing resolution $\Rightarrow$ local shortage/surplus of grid points
- **doesn't generalize to $>$ 1 spatial dimension (grid "tangles")**

[**Eulerian**] add/delete grid points as necessary, but don't move them
(at least not for mesh refinement)

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing ⇒ more complicated finite differencing
- rapidly changing resolution ⇒ local shortage/surplus of grid points
- **doesn't generalize to > 1 spatial dimension (grid "tangles")**

[**Eulerian**] add/delete grid points as necessary, but don't move them (at least not for mesh refinement)

- can arrange for finite differencing to see only uniform grid spacing
- can change resolution rapidly if necessary

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing $\Rightarrow$ more complicated finite differencing
- rapidly changing resolution $\Rightarrow$ local shortage/surplus of grid points
- **doesn't generalize to $>$ 1 spatial dimension (grid "tangles")**

[**Eulerian**] add/delete grid points as necessary, but don't move them
(at least not for mesh refinement)

- can arrange for finite differencing to see only uniform grid spacing
- can change resolution rapidly if necessary
- generalises cleanly to N spatial dimension

Typography of (Finite-Difference) Mesh Refinement

[**Lagrangian**] smoothly move grid points around to vary resolution

- relatively simple
- nonuniform grid spacing $\Rightarrow$ more complicated finite differencing
- rapidly changing resolution $\Rightarrow$ local shortage/surplus of grid points
- **doesn't generalize to $>$ 1 spatial dimension (grid "tangles")**

[**Eulerian**] add/delete grid points as necessary, but don't move them
(at least not for mesh refinement)

- can arrange for finite differencing to see only uniform grid spacing
- can change resolution rapidly if necessary
- generalises cleanly to N spatial dimension
- **relatively hard to implement**

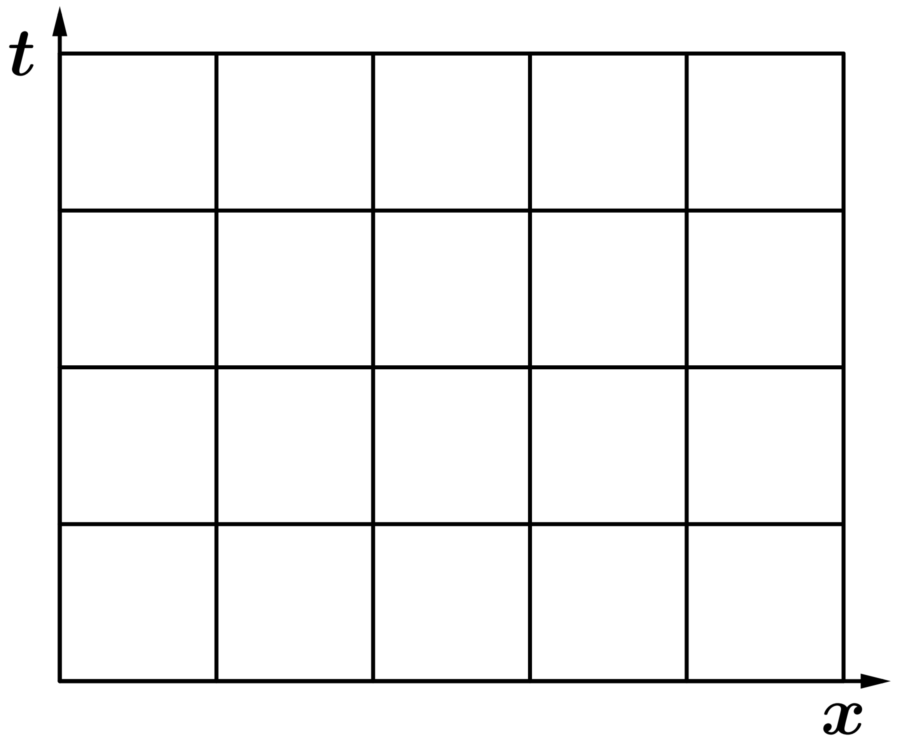
Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

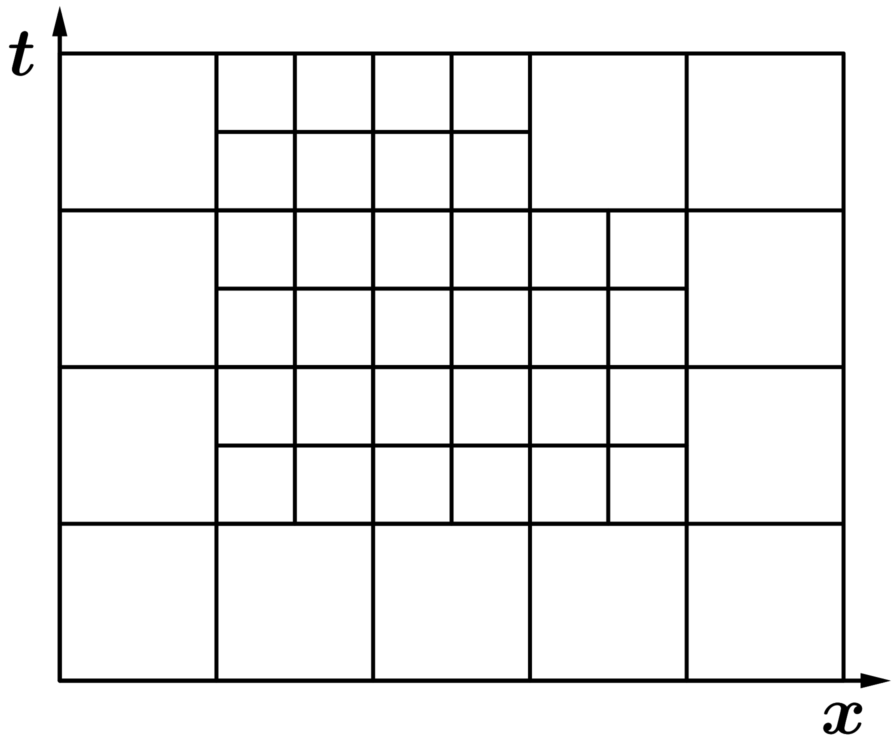
- use **locally uniform grids** (possibly in curvlinear coordinates)
- coarsest grid covers entire problem domain



Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

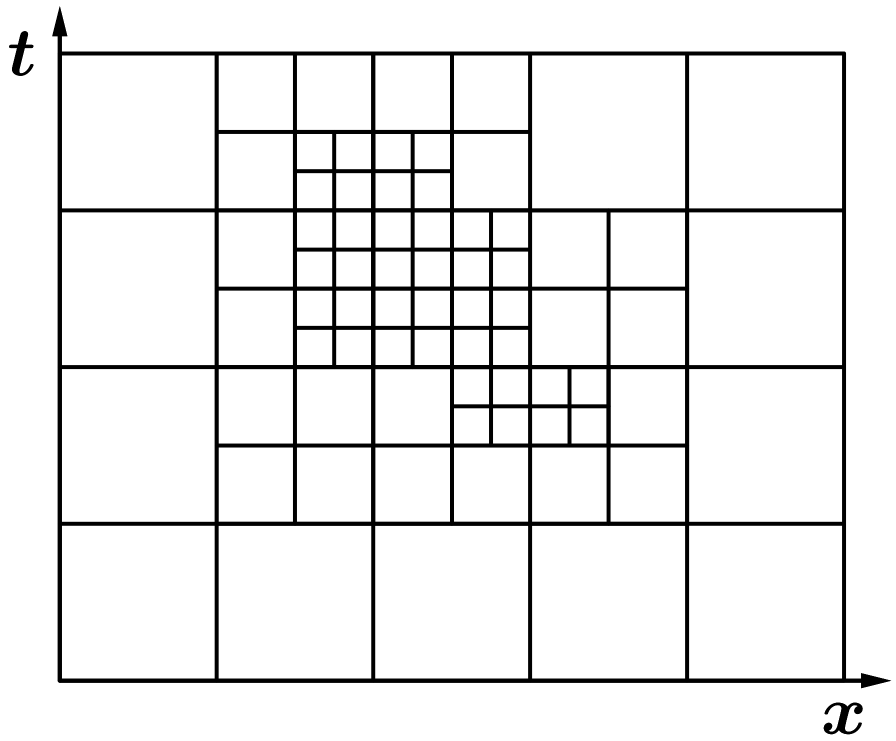
- use **locally uniform grids** (possibly in curvlinear coordinates)
- coarsest grid covers entire problem domain
- **refine in both space and time**



Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

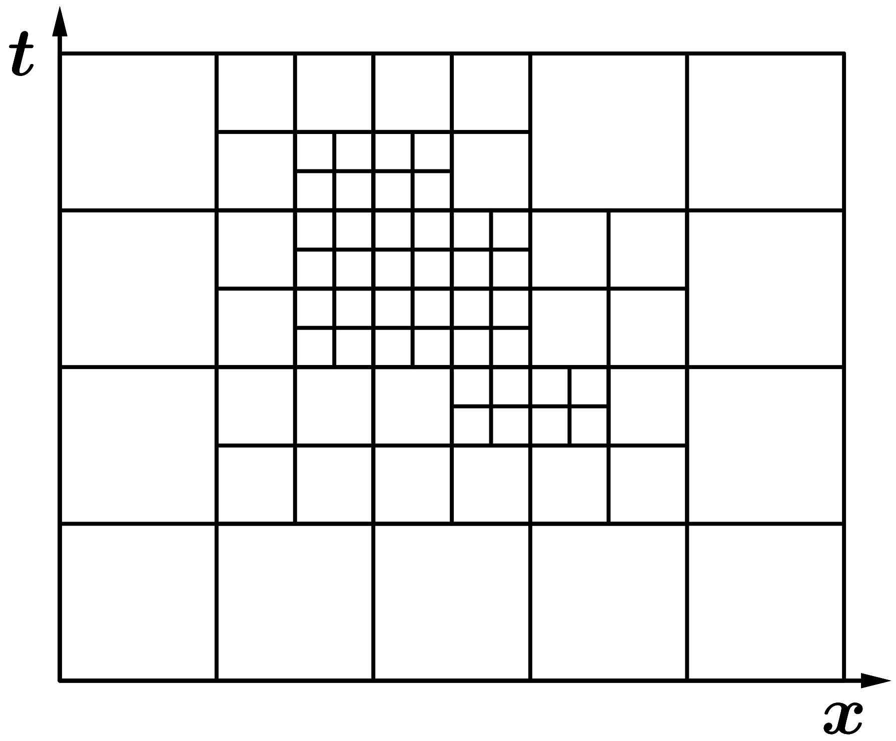
- use **locally uniform grids** (possibly in curvlinear coordinates)
- coarsest grid covers entire problem domain
- **refine in both space and time**
- fine grids **overlay** coarser grids



Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

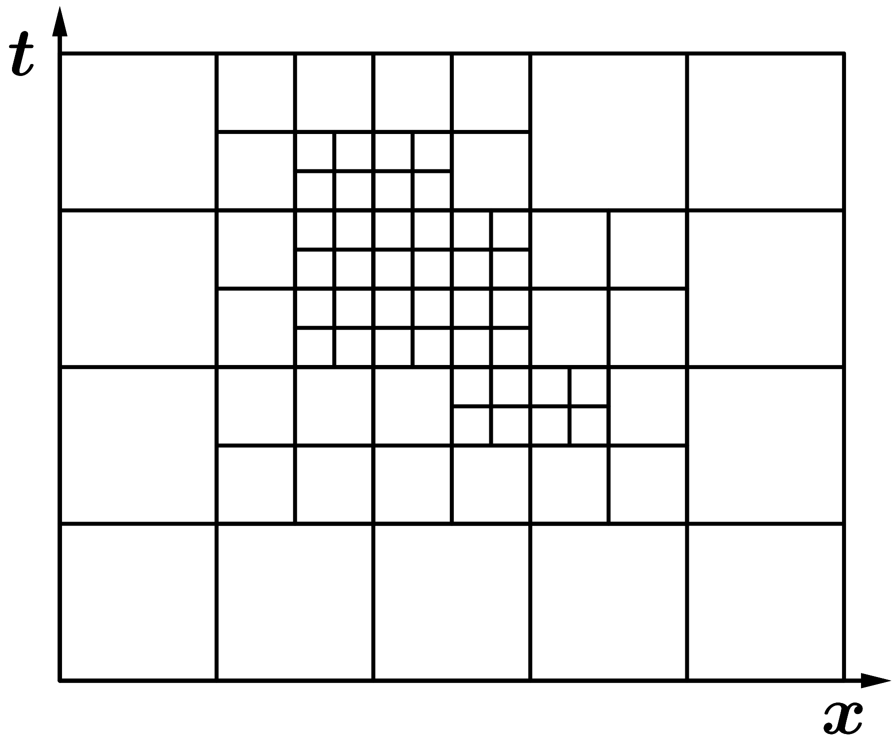
- use **locally uniform grids** (possibly in curvlinear coordinates)
- coarsest grid covers entire problem domain
- **refine in both space and time**
- fine grids **overlay** coarser grids
- fine-grid initial & boundary data is **interpolated** from coarse grids



Eulerian Mesh Refinement: Overview

Berger & Oliger ⟦*J. Comp. Phys.* **53**, 484 (1984)⟧ defined what has become the standard approach to Eulierian finite-difference mesh refinement for "hyperbolic-like" time-evolution PDEs:

- use **locally uniform grids** (possibly in curvlinear coordinates)
- coarsest grid covers entire problem domain
- **refine in both space and time**
- fine grids **overlay** coarser grids
- fine-grid initial & boundary data is **interpolated** from coarse grids
- integrate each grid independently
- inject fine results back into coarse grid when/where points coincide (keeps coarse grid accurate)



Software Approaches to Berger-Oliger

Berger-Oliger is conceptually fairly simple,

but there are a lot of messy details in an implementation.

Software Approaches to Berger-Oliger

Berger-Oliger is conceptually fairly simple,

but there are a lot of messy details in an implementation.

Some people use generic toolkits:

- AMRD/PAMR (Choptuik/Pretorius)
- DAGH/GRACE (Parashar)
- PARAMESH (MacNeice *et al.*, NASA/Goddard 2BH project)
- SAMRAI (Hornung *et al.*, Livermore)
- CARPET (Schnetter) (works as part of CACTUS)

$\Rightarrow$ relatively easy programming

but details may be undocumented, learning curves may be slow

Software Approaches to Berger-Oliger

Berger-Oliger is conceptually fairly simple,
but there are a lot of messy details in an implementation.

Some people use generic toolkits:

- AMRD/PAMR (Choptuik/Pretorius)
- DAGH/GrACE (Parashar)
- ParaMesh (MacNeice *et al.*, NASA/Goddard 2BH project)
- SAMRAI (Hornung *et al.*, Livermore)
- Carpet (Schnetter) (works as part of Cactus)

$\Rightarrow$ relatively easy programming
but details may be undocumented, learning curves may be slow

Another alternative is to write a Berger-Oliger code from scratch.
$\Rightarrow$ takes $\sim$5K–10K lines of code (including test drivers, comments, etc)

Berger-Oliger in Cauchy (Numerical) Relativity

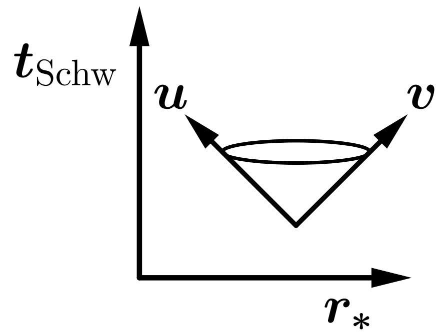
Choptuik pioneered the use of Berger-Oliger mesh refinement in numerical relativity in his discovery of self-similarity and critical phenomena in gravitational collapse. ⟦*Phys. Rev. Lett.* **70**, 9 (1993)⟧

Berger-Oliger methods are now widely used in $3 + 1$ numerical relativity. Some good references include:

- Berger & Oliger, *J. Comp. Phys.* **53**, 484 (1984)
- Berger, *SIAM J. Sci. Stat. Comput.* **7**, 904 (1986)
- Choptuik, "Experiences with an Adaptive Mesh Refinement Algorithm in Numerical Relativity", pages 206–221 in Evans, Finn, and Hobill, *Frontiers in Numerical Relativity*, Cambridge U.P., 1989
- Schnetter, Hawley, and Hawke, *Class. Quant. Grav.* **21**, 1465 (2004) [Nicely explains the complications which arise when equations contain 1st time derivatives but 2nd spatial derivatives.]

Characteristic Coordinates

Motivation: null boundaries are <u>much</u> nicer than timelike boundaries (both analytically and numerically)

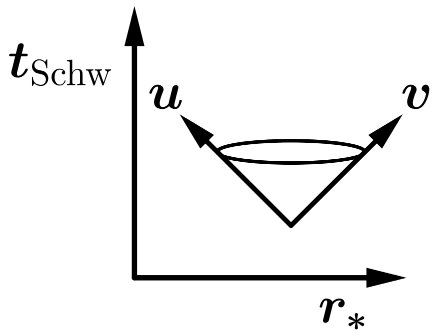


Null coordinates:

$$u = t_{\mathrm{Schw}} - r_*$$
$$v = t_{\mathrm{Schw}} + r_*$$

Characteristic Coordinates

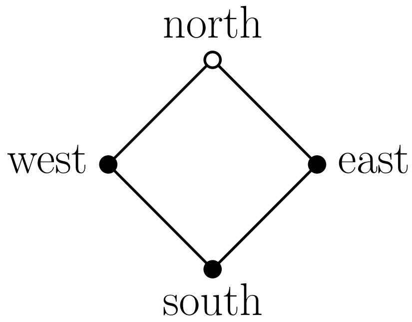
Motivation: null boundaries are <u>much</u> nicer than timelike boundaries (both analytically and numerically)



Null coordinates: $u = t_{\rm Schw} - r_*$

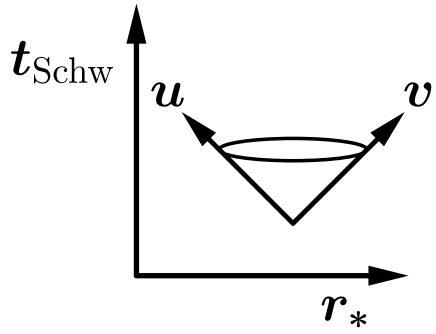
$$v = t_{\rm Schw} + r_*$$

Fundamental discretization uses double-null "diamond" cells



Characteristic Coordinates

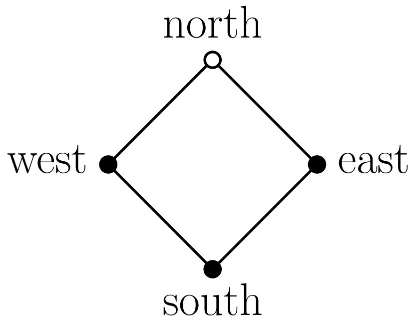
Motivation: null boundaries are <u>much</u> nicer than timelike boundaries (both analytically and numerically)



Null coordinates: $u = t_{\mathrm{Schw}} - r_*$

$$v = t_{\mathrm{Schw}} + r_*$$

Fundamental discretization uses double-null "diamond" cells

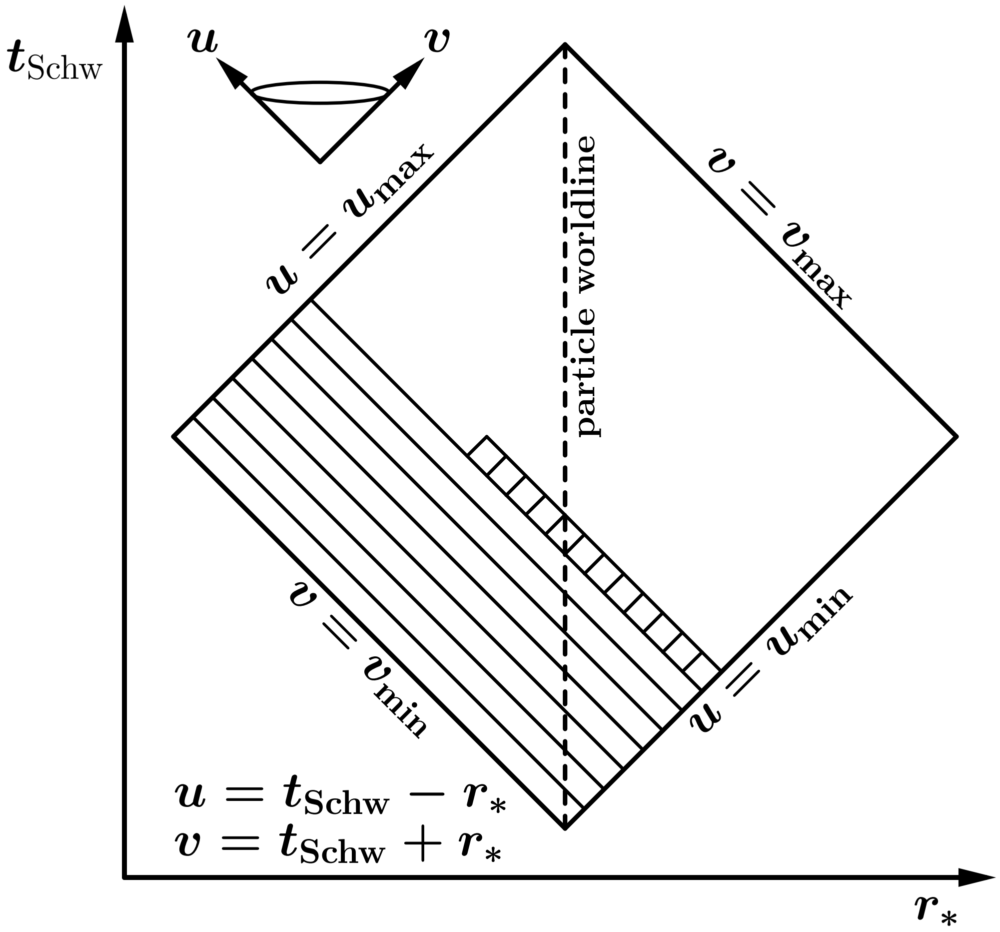


Integrate metric-perturbation equations over cell ⇒

$$\begin{aligned}\phi_{\mathrm{north}} \;&=\; \phi_{\mathrm{west}} + \phi_{\mathrm{east}} - \phi_{\mathrm{south}} - h^2 \frac{\phi_E + \phi_W}{2} V_{\ell_{\mathrm{center}}} \\ &\quad + h \operatorname{sinc}(\tfrac{1}{2} m \omega_{\mathrm{orbit}} h) S_{\ell m}(t_{\mathrm{center}}) \quad \text{[only for particle in cell]} \\ &\quad + \mathcal{O}(h^{-4}) \text{ [vacuum] or } \mathcal{O}(h^{-3}) \text{ [particle]}\end{aligned}$$

Self-Force Calculations (Characteristic Coordinates)

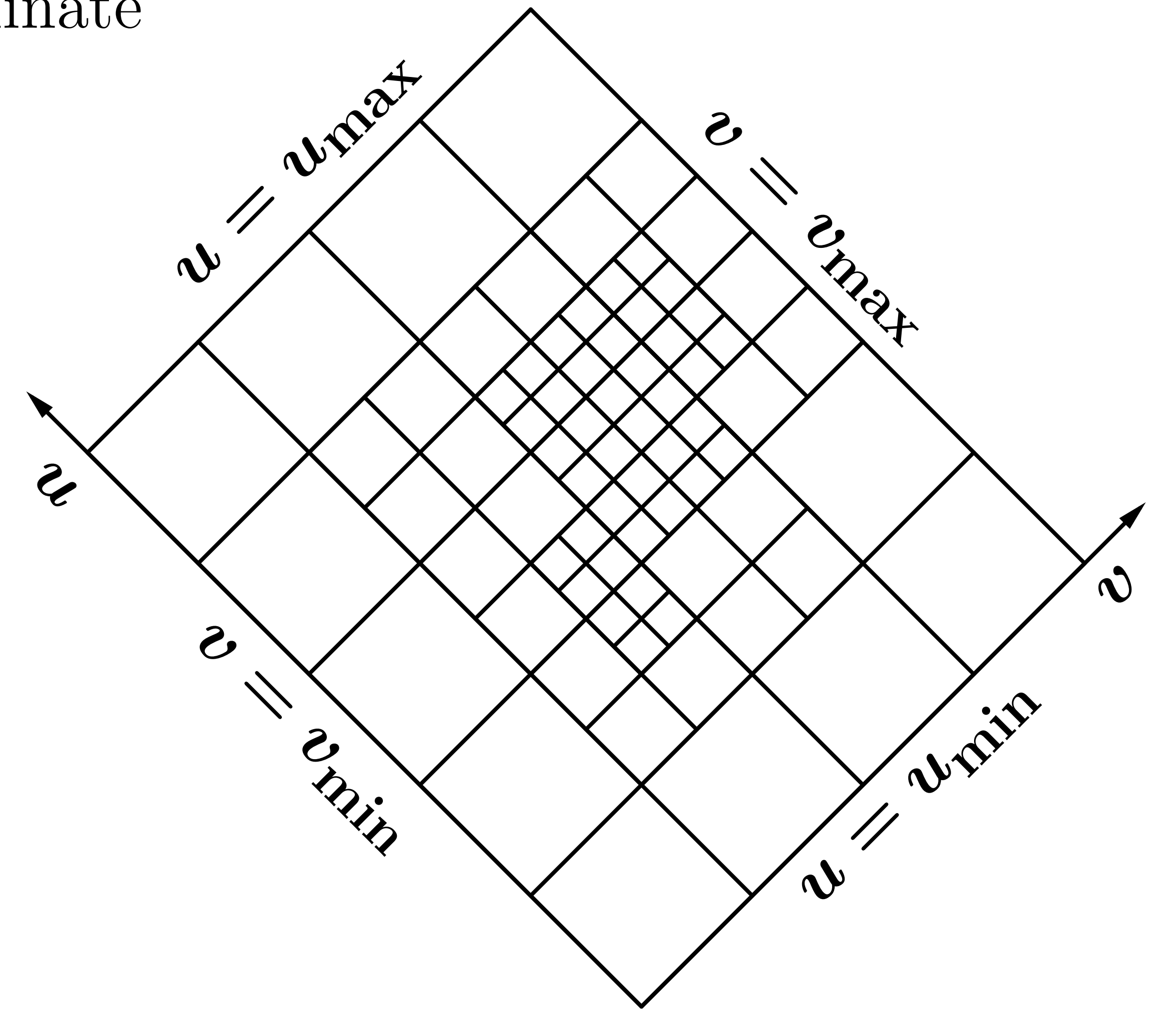
As discussed by Leor Barack & Norichika Sago in their talks, for self-force calculations we integrate the (discretized) metric perturbation equations for each (ℓ, m) multipole mode, in a square "grid box" in (u, v) space, chosen to be big enough for the initial-data field perturbations to have decayed below our numerical error levels by the end of the integration.



The integration proceeds one $v =$ constant slice at a time; each slice is integrated one diamond cell at a time.

Berger-Oliger in Characteristic Coordinates

Basically, use standard Berger-Oliger mesh refinement, treating u as a "spatial" coordinate on $v =$ constant slices, and v as a "time" coordinate



Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

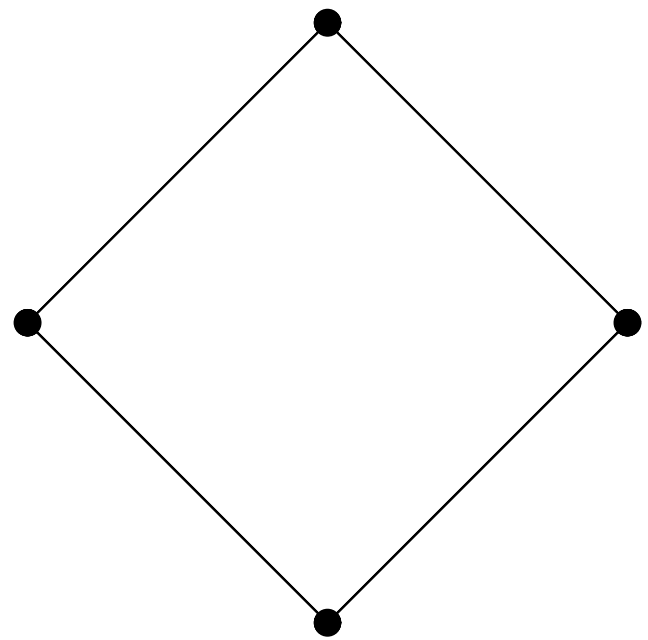
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell



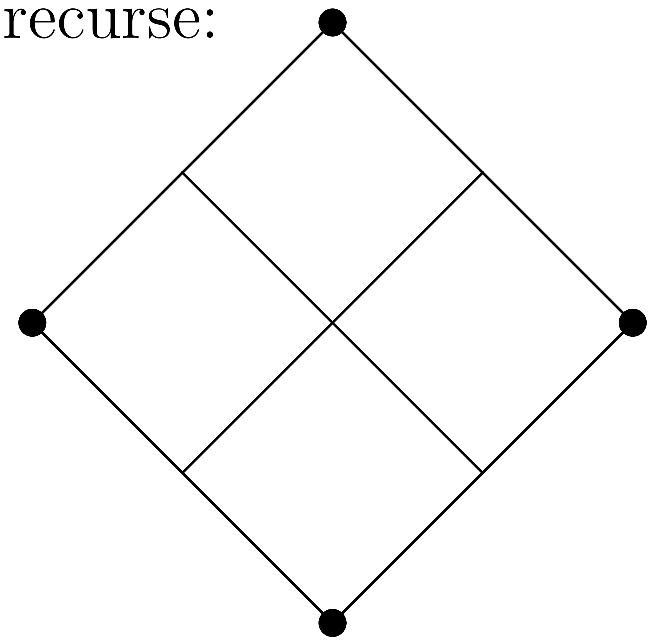
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells



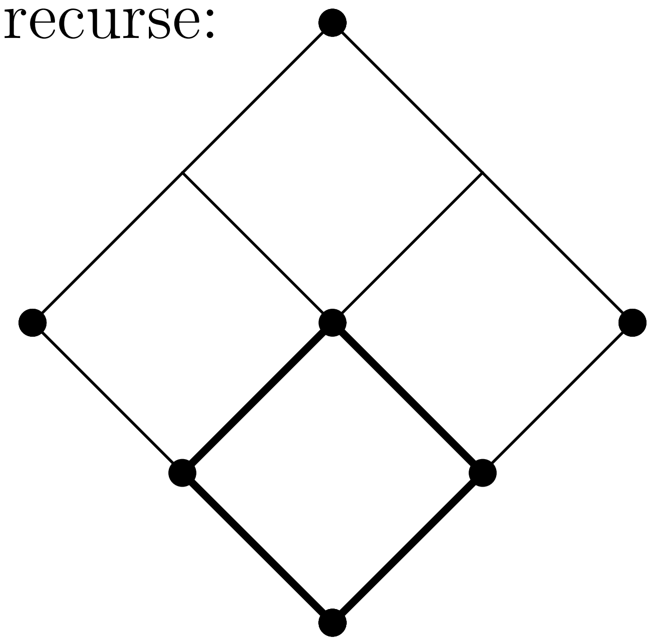
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell



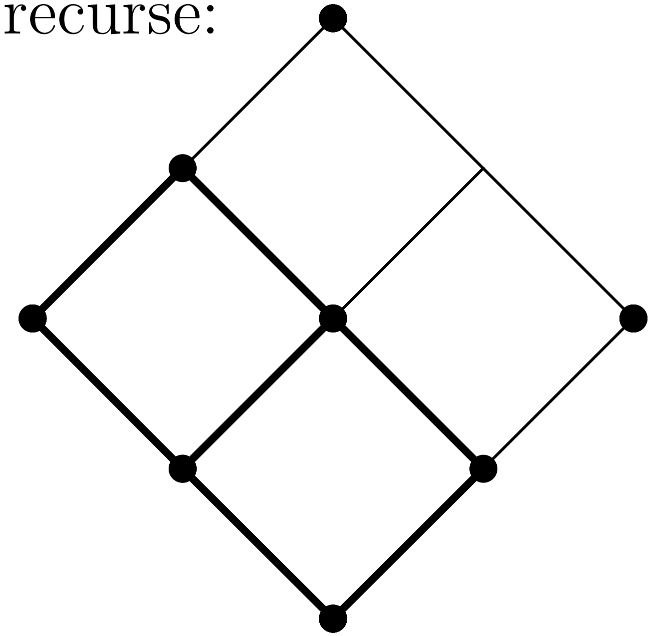
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell
  - recursively integrate west subcell



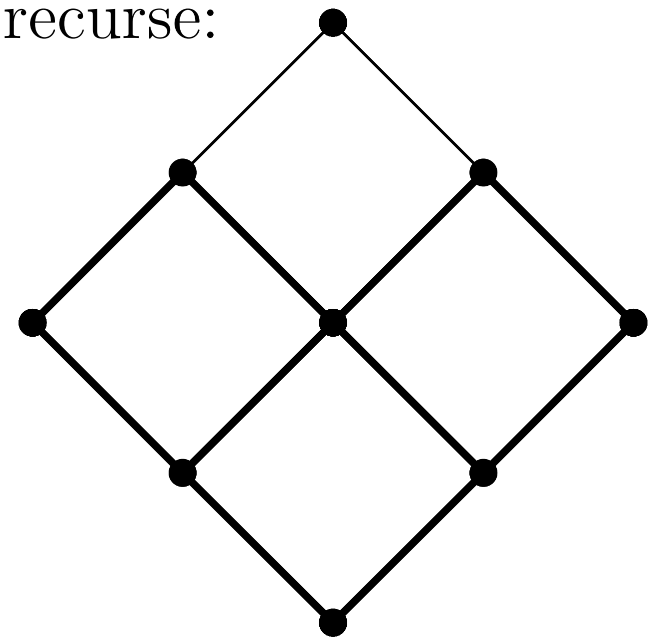
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell
  - recursively integrate west subcell
  - recursively integrate east subcell



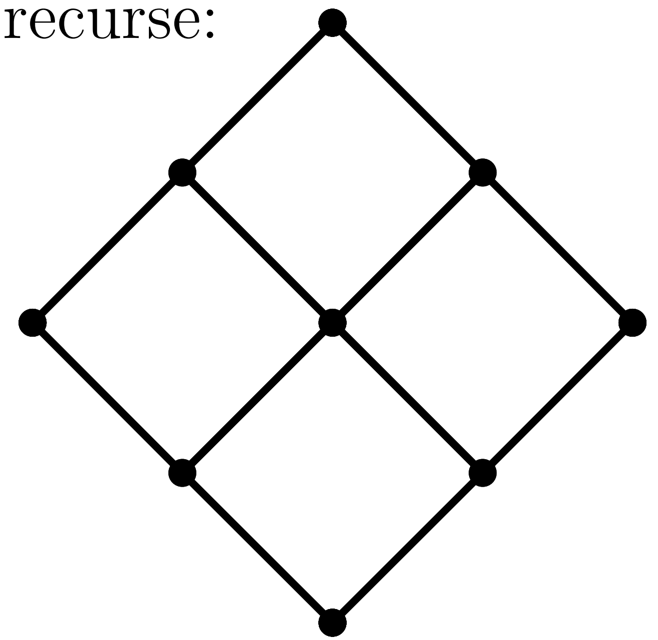
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell
  - recursively integrate west subcell
  - recursively integrate east subcell
  - recursively integrate north subcell



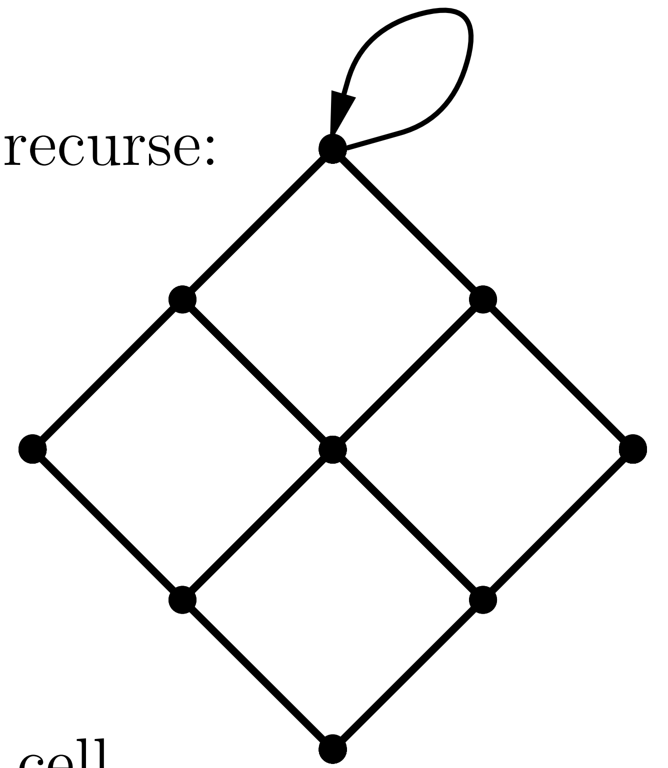
Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧

- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell
  - recursively integrate west subcell
  - recursively integrate east subcell
  - recursively integrate north subcell
  - inject north subcell result back into coarser cell

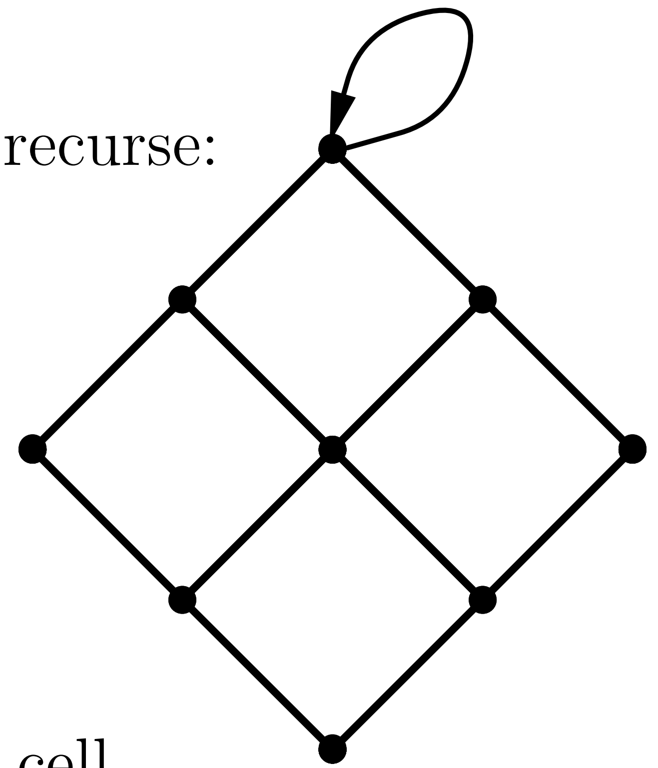


Berger-Oliger in Characteristic Coordinates (2)

Cell-recursive algorithm:

⟦Hamadé & Stewart, *Class. Quant. Grav.* **13**, 497 (1996)⟧

⟦Pretorius & Lehner, *J. Comp. Phys.* **198**, 10 (2004) = gr-qc/0302003⟧



- integrate a cell
- if local truncation error estimate is too large, recurse:
  - divide cell into 4 subcells
  - recursively integrate south subcell
  - recursively integrate west subcell
  - recursively integrate east subcell
  - recursively integrate north subcell
  - inject north subcell result back into coarser cell

Problem: fine-grained (per-cell) memory management

Slice-Recursive Berger-Oliger Mesh Refinement

Fundamental concept: recurse on entire $v = \text{constant}$ slices

Slice-Recursive Berger-Oliger Mesh Refinement

Fundamental concept: recurse on entire v = constant slices

- **integrate an entire v = constant slice**, flagging points where local truncation error estimate > threshold

Slice-Recursive Berger-Oliger Mesh Refinement

Fundamental concept: recurse on entire $v =$ constant slices

- **integrate an entire $v =$ constant slice**, flagging points where local truncation error estimate > threshold
- if there are flagged points, recurse:
  - divide slice into 2 subslices
  - recursively integrate smaller-v ("lower") subslice
  - recursively integrate larger-v ("upper") subslice
  - inject upper-slice results back into matching points of original (coarse) slice
  - reintegrate the remainder of the original (coarse) slice starting from the upper-most injected data

Slice-Recursive Berger-Oliger Mesh Refinement

Fundamental concept: recurse on entire v = constant slices

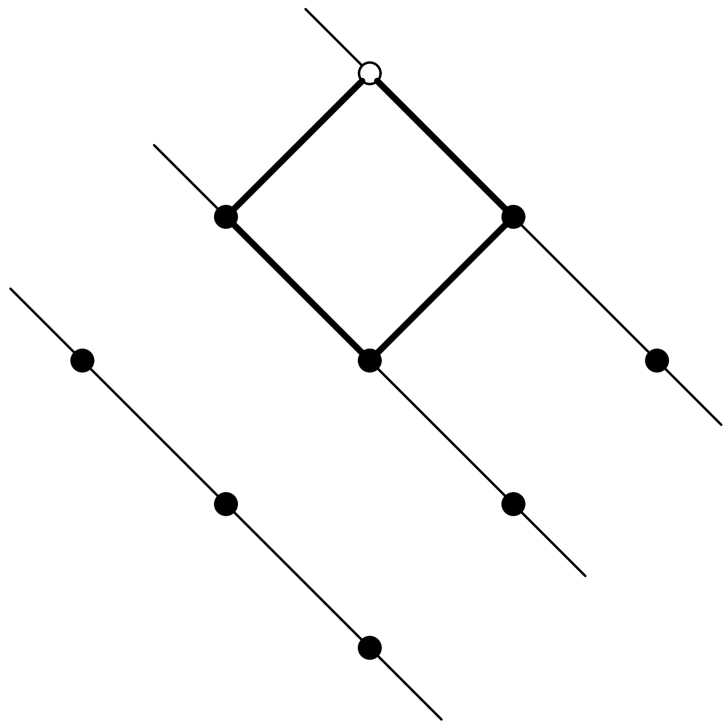
- **integrate an entire v = constant slice**, flagging points where local truncation error estimate > threshold
- if there are flagged points, recurse:
  - divide slice into 2 subslices
  - recursively integrate smaller-v ("lower") subslice
  - recursively integrate larger-v ("upper") subslice
  - inject upper-slice results back into matching points of original (coarse) slice
  - reintegrate the remainder of the original (coarse) slice starting from the upper-most injected data

⇒ Relatively simple bookkeeping & memory management

Error Estimation for Adaptive Mesh Refinement

For **adaptive** mesh refinement (AMR), we need to estimate (during the integration) when the integration is accurate enough, and when it's not. To do this, we use an estimate of the local finite differencing error, also known as the local truncation error (LTE): compare

(a) the result of integrating a standard diamond cell

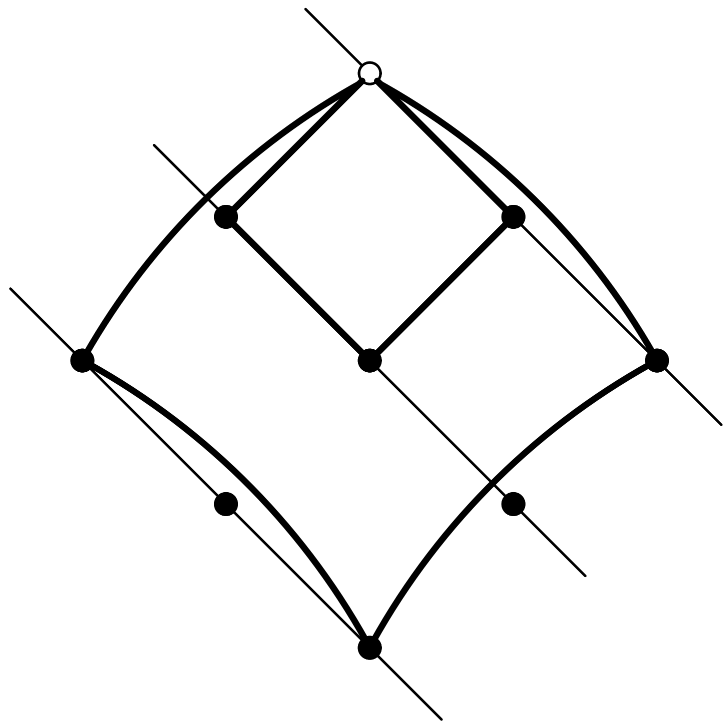


Error Estimation for Adaptive Mesh Refinement

For **adaptive** mesh refinement (AMR), we need to estimate (during the integration) when the integration is accurate enough, and when it's not. To do this, we use an estimate of the local finite differencing error, also known as the local truncation error (LTE): compare

(a) the result of integrating a standard diamond cell

(b) the result of integrating a double-sized diamond cell using data from the 2nd-to-last and current slices



The difference (b) − (a) is an estimate of the LTE, up to some $\mathcal{O}(1)$ factor; the AMR refinement criterion is simply

if $\left(\left|(b) - (a)\right| > \text{threshold}\right)$

then this cell must be redone at higher resolution

Test Cases for Sample Results

Physics:

- Schwarzschild background, scalar particle in circular orbit
- zero initial data on $v = v_{\min}$ and $u = u_{\min}$ grid-box faces

Test Cases for Sample Results

Physics:

- Schwarzschild background, scalar particle in circular orbit
- zero initial data on $v = v_{\text{min}}$ and $u = u_{\text{min}}$ grid-box faces

Numerical Methods:

- 2nd order finite differencing (global accuracy)
- **slice-recursive AMR algorithm**
- AMR gradually turned starting at $100m$ (we don't bother resolving the junk radiation at the start of the evolution); AMR fully active by $\approx 200m$

Sample Results (Convergence Tests)

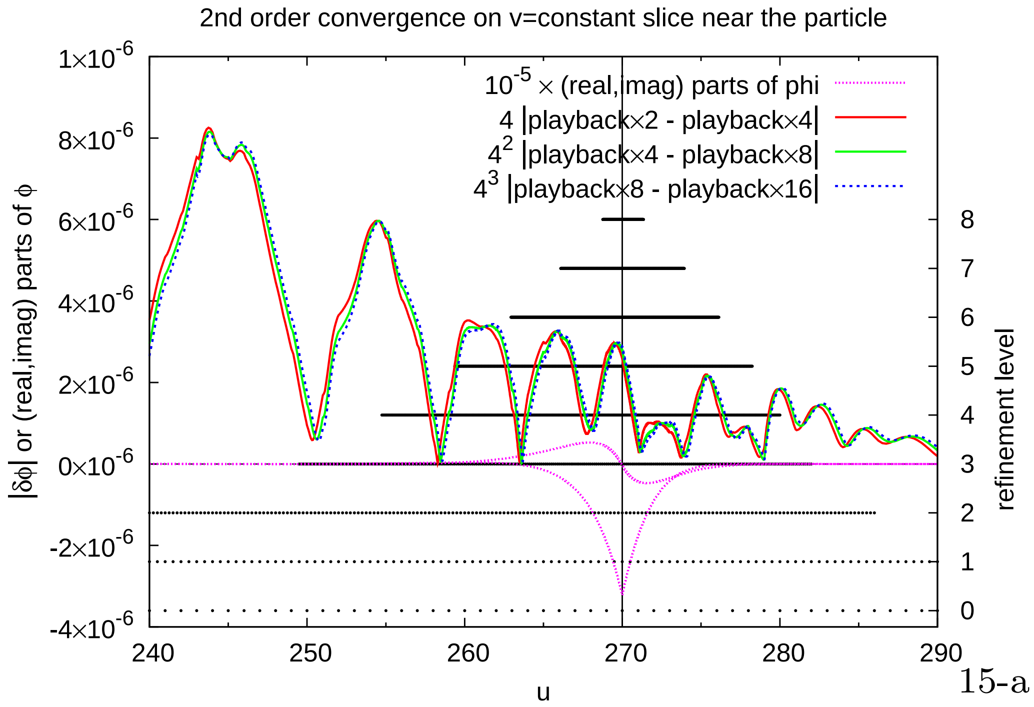
To test convergence, the code writes out a “script” describing the mesh-refinement structure, which can then be “played back” at successively higher resolutions.

Sample Results (Convergence Tests)

To test convergence, the code writes out a "script" describing the mesh-refinement structure, which can then be "played back" at successively higher resolutions.

Test case: $(\ell, m) = (10, 10)$, particle at $r_* = 10m$ ($r_{\text{Schw}} = 7.85m$)
script "recorded" at: AMR error threshold 10^{-6} in $|\phi|$
(coarsest grid resolution $0.5m$)

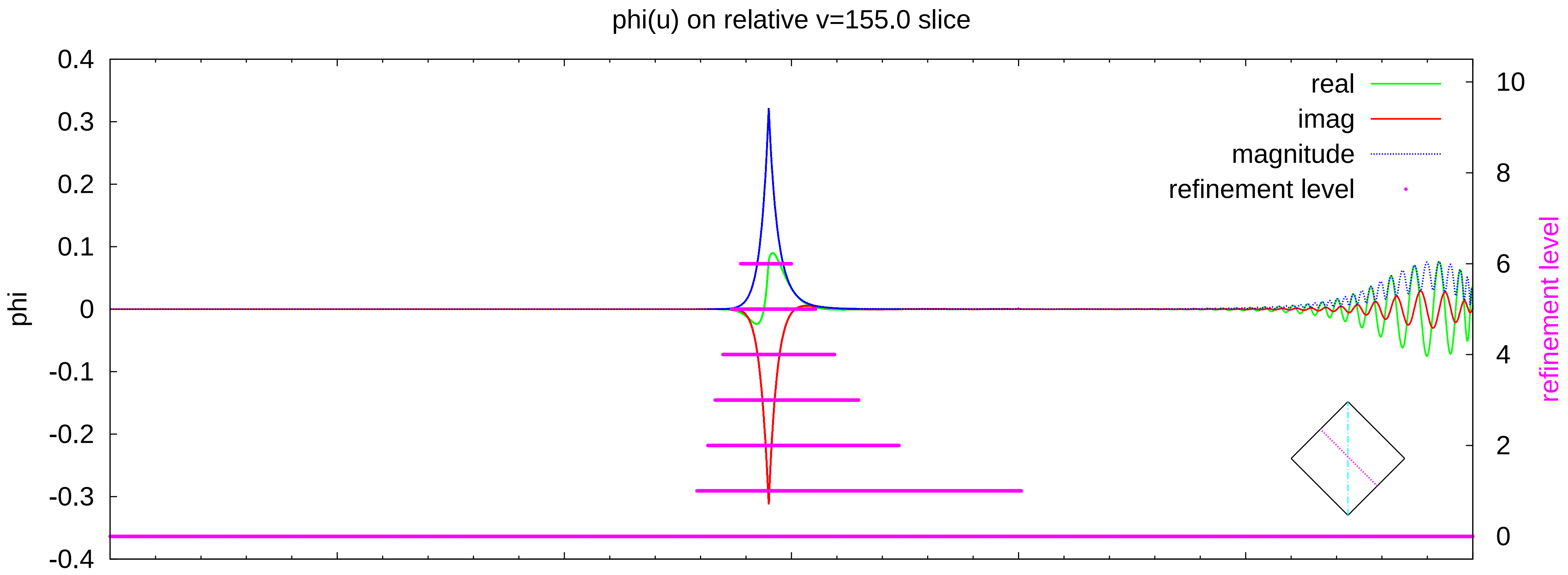
The code shows excellent 2nd order convergence, even across the mesh-refinement boundaries and near the particle worldline, (where $\partial_r\phi$ has a jump discontinuity)



Sample Results (ϕ movie)

Test case: same as before, but error threshold 10^{-7}
(coarsest grid resolution $0.1m$)

Sample frame from **movie** (see also **poster** outside)



Sample Results (Self-Force)

Test case:

- particle at $r_{\text{Schw}} = 10m$
- modes computed numerically for $\ell \leq 20$
  (121 modes given even/odd symmetry)
- F_ℓ for $\ell > 20$ tail estimated via 3-term fit to $\{\ell^{-2}, \ell^{-4}, \ell^{-6}, \ldots\}$ series
  ⟦Detweiler, Messaritaki, & Whiting, *Phys. Rev. D* **67**, 104016 (2003)⟧
- grid box $300m$ on a side (too small)
- AMR error threshold 10^{-6} in $|\phi|$, Richardson extrap. playback$\times\{6, 8\}$
- error estimate via $|F^r_{\text{inside}} - F^r_{\text{outside}}|$

Sample Results (Self-Force)

Test case:

- particle at $r_{\text{Schw}} = 10m$
- modes computed numerically for $\ell \leq 20$
  (121 modes given even/odd symmetry)
- F_ℓ for $\ell > 20$ tail estimated via 3-term fit to $\{\ell^{-2}, \ell^{-4}, \ell^{-6}, \dots\}$ series
  ⟦Detweiler, Messaritaki, & Whiting, *Phys. Rev. D* **67**, 104016 (2003)⟧
- grid box $300m$ on a side (too small)
- AMR error threshold 10^{-6} in $|\phi|$, Richardson extrap. playback$\times\{6, 8\}$
- error estimate via $|F^r_{\text{inside}} - F^r_{\text{outside}}|$

$\Rightarrow$	this work	$F^r = 1.3808 \times 10^{-5}$ (43% numerical, 57% tail)
	Detweiler *et al.*	$F^r = 1.3784 \times 10^{-5}$
	fractional error	0.17%

Conclusions

Adaptive Mesh Refinement (AMR):

- works well (big efficiency/accuracy gains)
- characteristic coordinates

  $\Rightarrow$ only small changes to Cauchy Berger-Oliger techniques/codes

Conclusions

Adaptive Mesh Refinement (AMR):

- works well (big efficiency/accuracy gains)
- characteristic coordinates
  $\Rightarrow$ only small changes to Cauchy Berger-Oliger techniques/codes
- programming is complicated

Conclusions

Adaptive Mesh Refinement (AMR):

- works well (big efficiency/accuracy gains)
- characteristic coordinates
  $\Rightarrow$ only small changes to Cauchy Berger-Oliger techniques/codes
- programming is complicated

Self-Force:

- nice results for Schwarzschild background,
  scalar particle in circular orbit

Conclusions

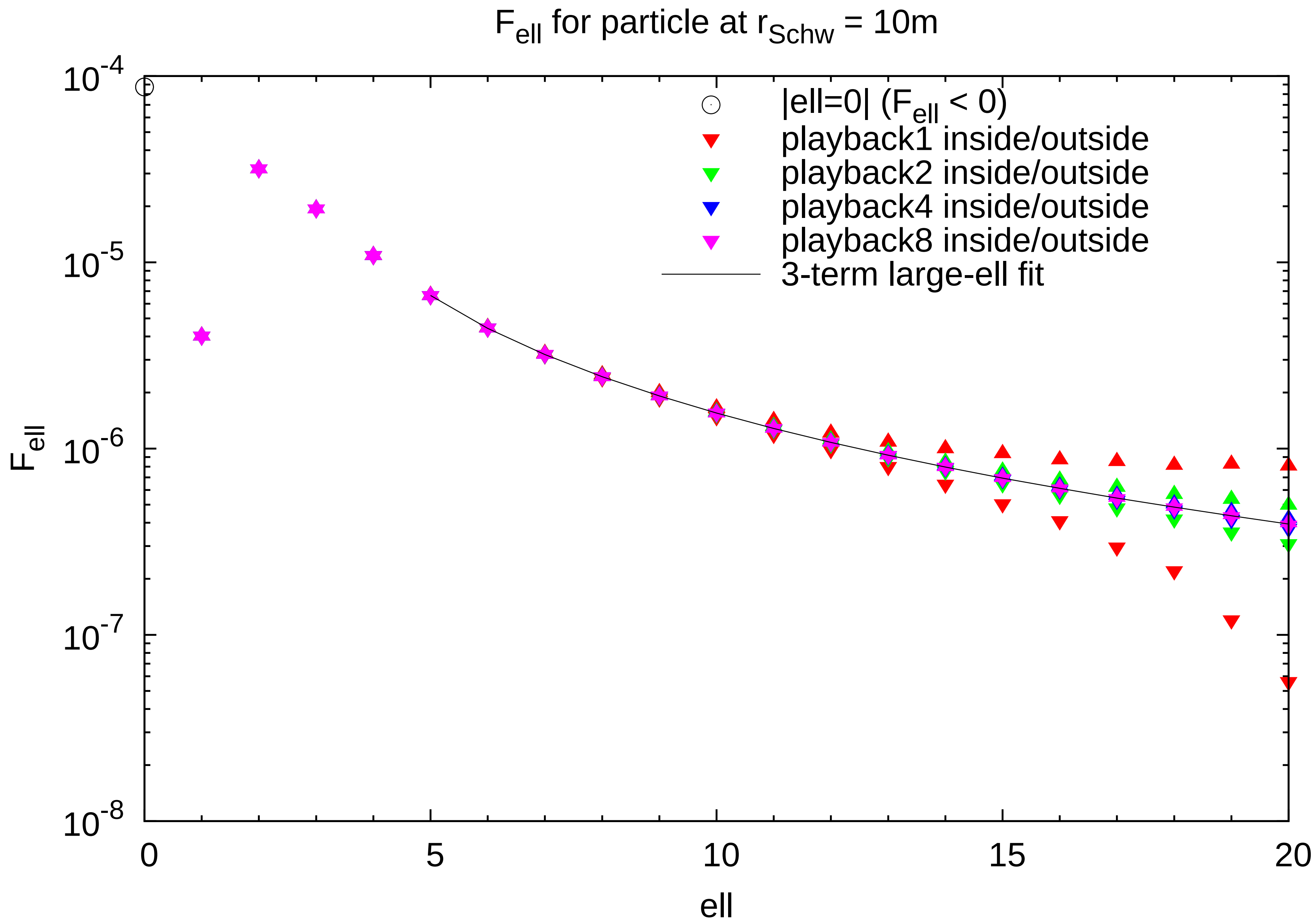
Adaptive Mesh Refinement (AMR):

- works well (big efficiency/accuracy gains)
- characteristic coordinates
  $\Rightarrow$ only small changes to Cauchy Berger-Oliger techniques/codes
- programming is complicated

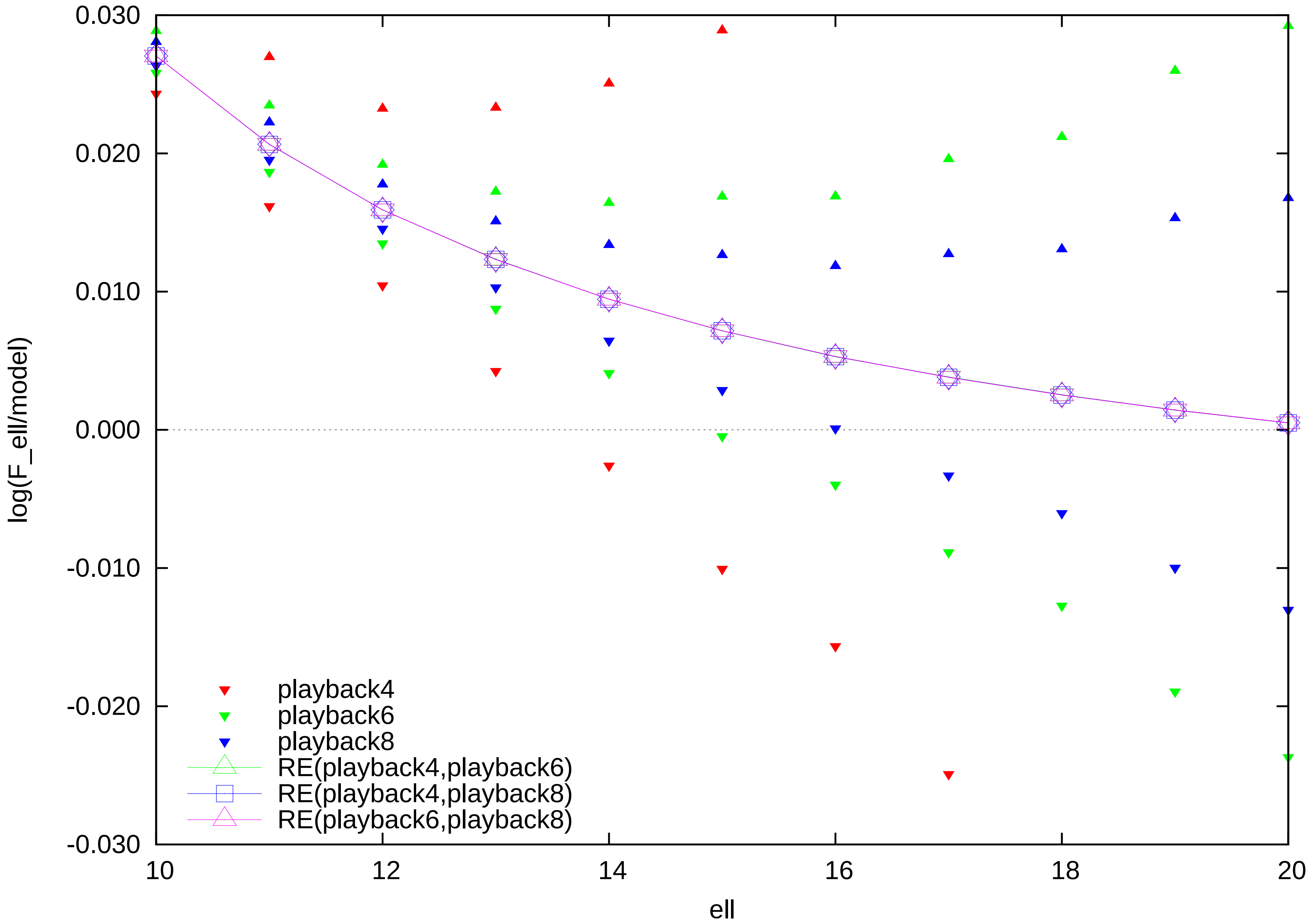
Self-Force:

- nice results for Schwarzschild background,
  scalar particle in circular orbit
- next steps: **4th order, eccentric orbits, Kerr**

Self-Force Results ($F_\ell(\ell)$)

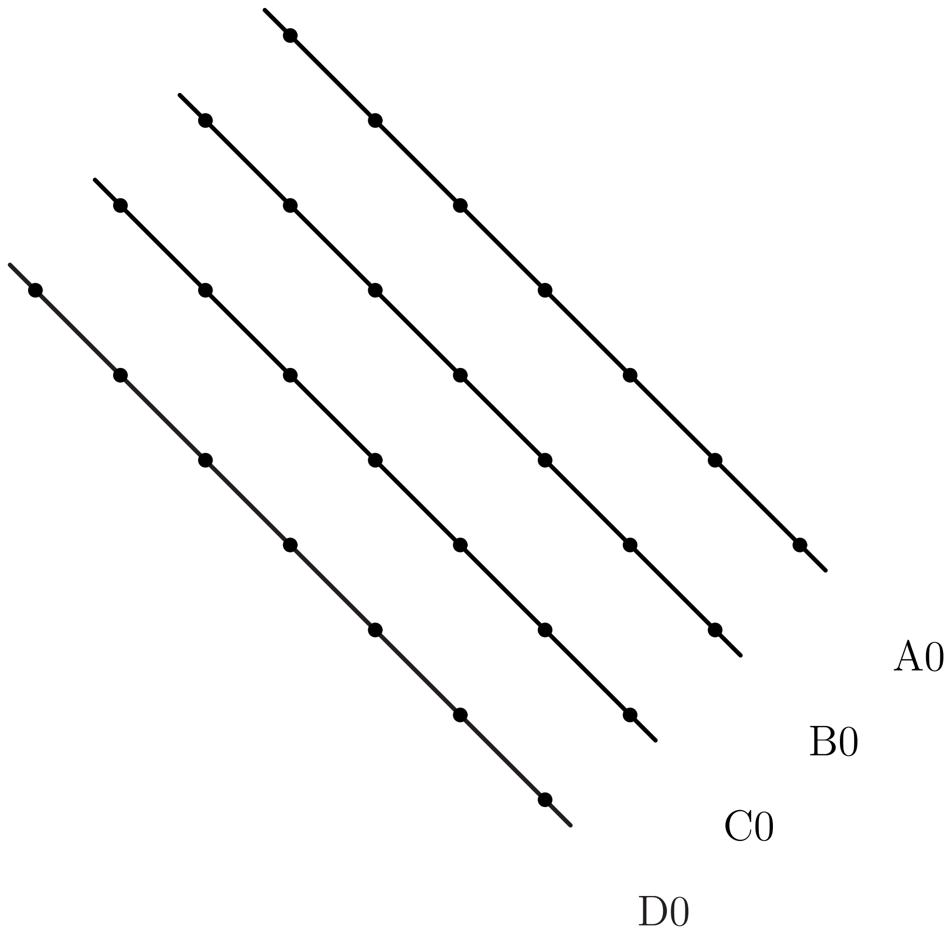


Self-Force Results (scaled $F_\ell(\ell)$)



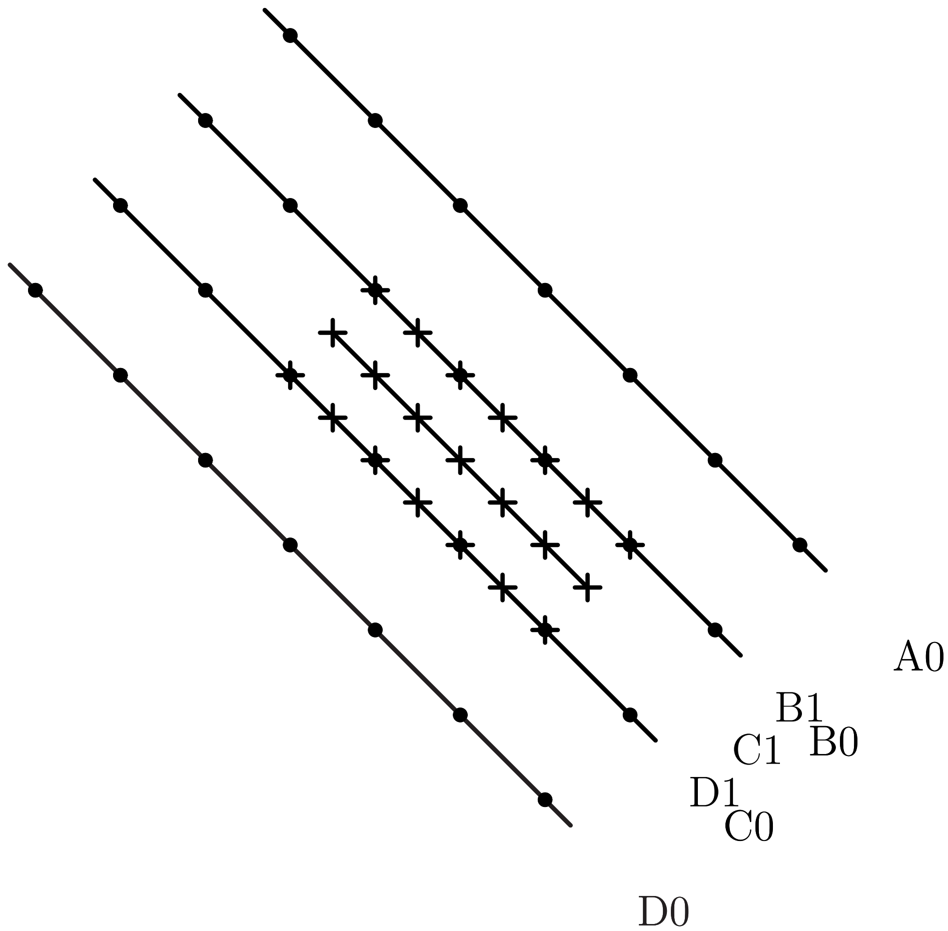
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)



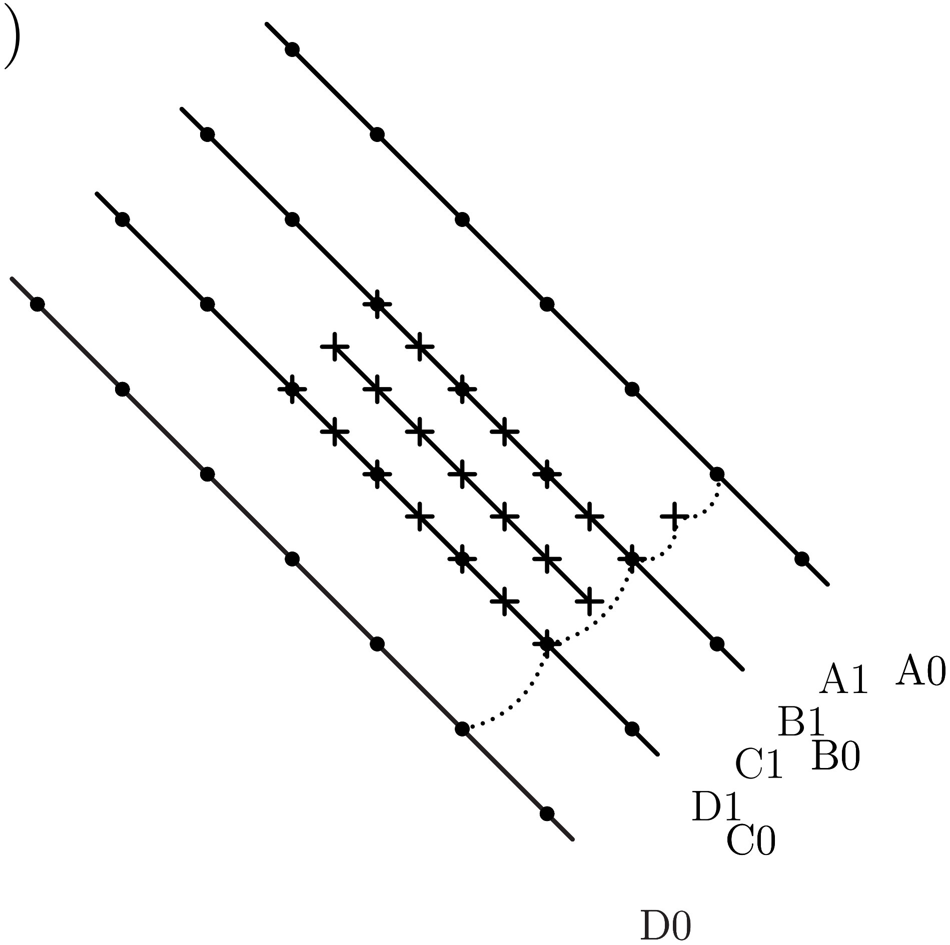
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region



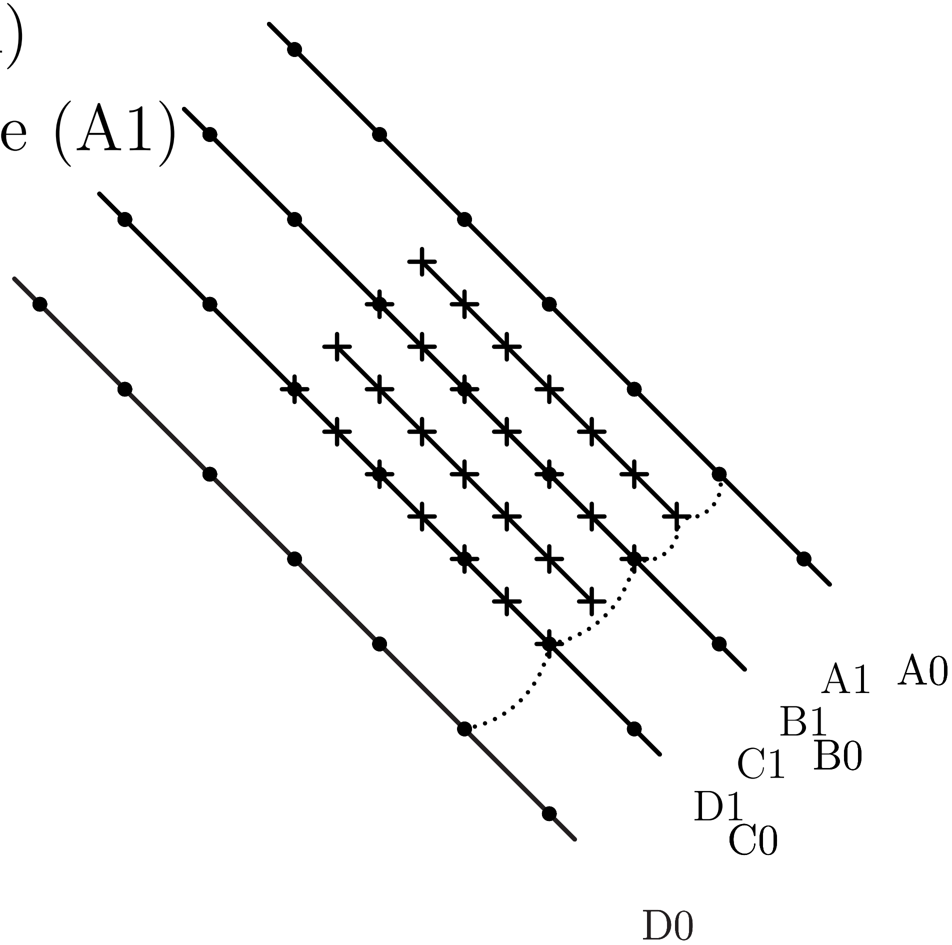
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting ($u_{\rm min}$) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)



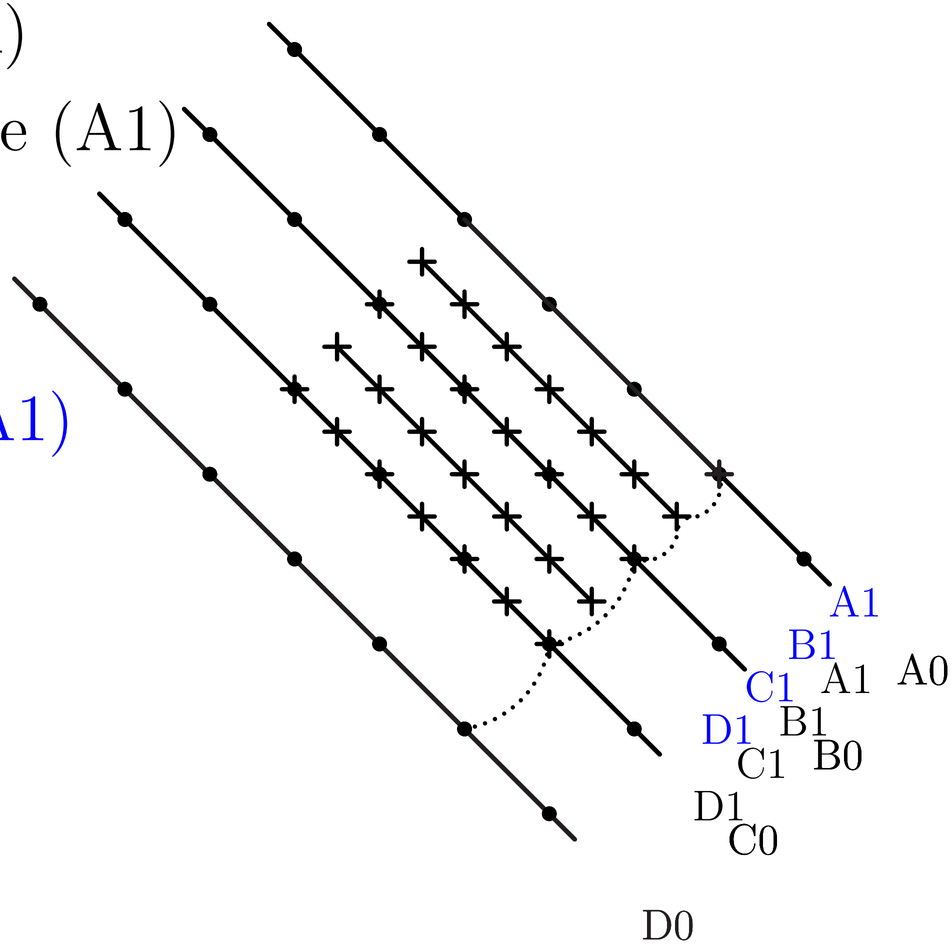
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting ($u_{\min}$) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)
- recursively integrate **entire** fine slice (A1)



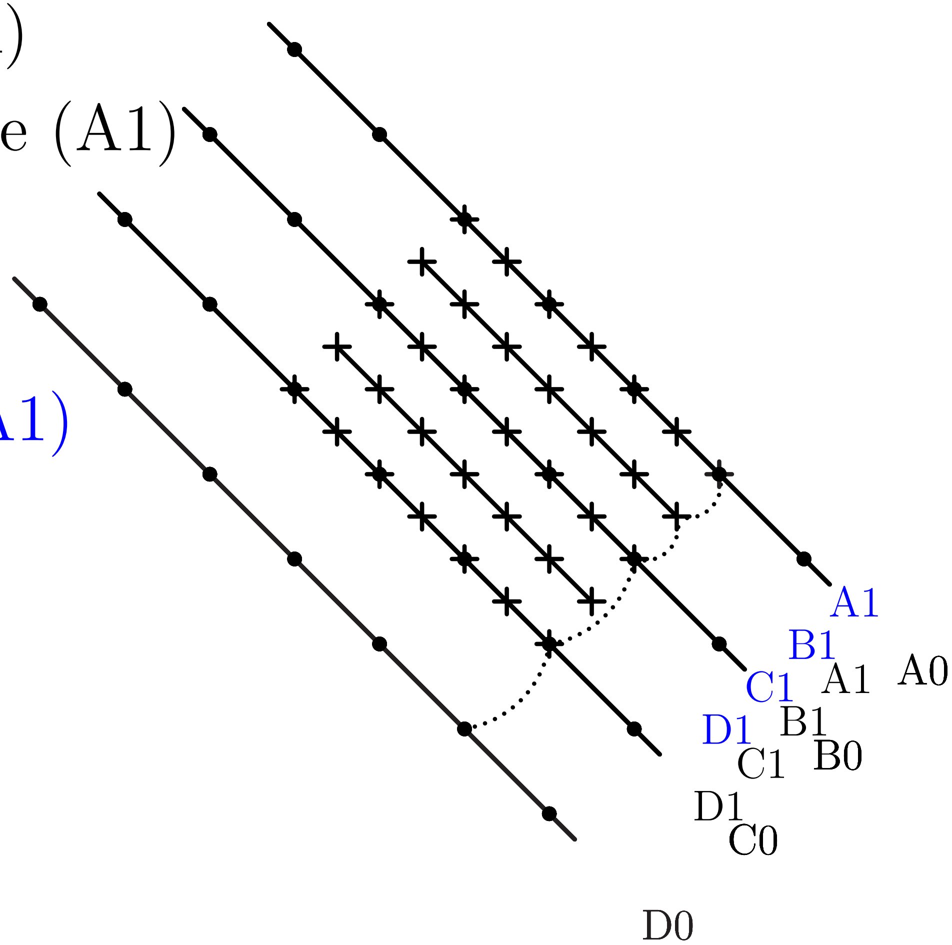
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting ($u_{\min}$) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)
- recursively integrate **entire** fine slice (A1)
- rotate fine slices
- copy coarse slice (A1) value to get starting ($u_{\min}$) value for fine slice (A1)



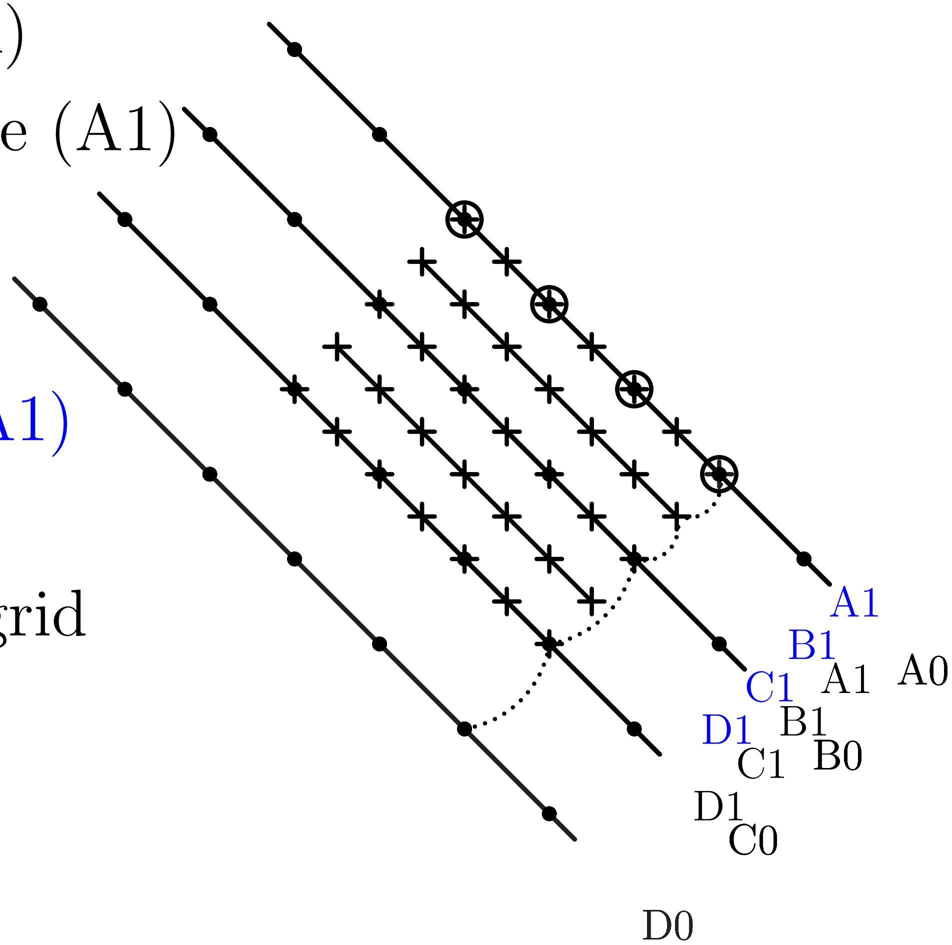
Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting ($u_{\min}$) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)
- recursively integrate **entire** fine slice (A1)
- rotate fine slices
- copy coarse slice (A1) value to get starting ($u_{\min}$) value for fine slice (A1)
- integrate **entire** fine slice (A1)



Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting (u_{min}) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)
- recursively integrate **entire** fine slice (A1)
- rotate fine slices
- copy coarse slice (A1) value to get starting (u_{min}) value for fine slice (A1)
- integrate **entire** fine slice (A1)
- copy fine slice (A1) back to coarse grid where grid points coincide



Slice-Recursive Berger-Oliger Mesh Refinement

- integrate **entire** coarse slice (A0)
- flag points if error estimate > threshold; fine grid ≈ flagged region
- time-interpolate to get starting ($u_{\min}$) value for fine slice (A1) (4 time levels ⇒ cubic interpolation)
- recursively integrate **entire** fine slice (A1)
- rotate fine slices
- copy coarse slice (A1) value to get starting ($u_{\min}$) value for fine slice (A1)
- integrate **entire** fine slice (A1)
- copy fine slice (A1) back to coarse grid where grid points coincide
- re-integrate remainder of coarse slice with updated "starting" values