First Common Refinement Implementation

Dan Ibanez

August 18, 2016

Abstract

This document describes how we converted the R3D library for Alexa use, implemented common refinement transfer for element density, made a correction for the case when boundaries change, and explains challenges to implementing momentum-conserving velocity remap.

1 Modifications to R3D

The R3D library by Devon Powell at Los Alamos (`https://github.com/devonmpowell/r3d`) provides the following operations:

1. "clip": intersect a polyhedron with a half-space, equivalent to clipping it by a plane.

2. "reduce": integrate a polynomial of *arbitrary order* over a polyhedron.

Since tetrahedra are polyhedra that can be described as the intersection of four half-spaces, the "clip" operation can be repeated four times to implement the intersection of two tetrahedra.

The output of the "reduce" function is an array of moments, which as I understand it are each some integral:

$$\int_V x^p y^q z^r \, \mathrm{d}V \tag{1}$$

They can be multiplied by polynomial coefficients to get the final integral value, for example:

$$\int_\Omega a_0 + a_1 x + a_2 y + a_3 z \, \mathrm{d}V = a_0 \int_\Omega 1 \, \mathrm{d}V + a_1 \int_\Omega x \, \mathrm{d}V + a_2 \int_\Omega y \, \mathrm{d}V + a_3 \int_\Omega z \, \mathrm{d}V \tag{2}$$

The original implementation also has the following key benefits:

1. polyedra are stored in fixed-size arrays (this allows us to use functional programming which assists optimization)

2. none of the code dynamically allocates memory (this is required for execution on GPUs and performance on CPUs)

3. the intersection of two tetrahedra can be performed in under 10 microseconds on a typical CPU.

4. the code is small enough to enable the transformations we made to it

We copied it into Omega_h because it needed a few key transformations to fit our needs:

1. use the C++11 language instead of C (this is just for easy compatibility with the rest of Alexa and Omega_h, and to remove certain GPU-incompatible function calls like "memset").

2. annotate the functions with KOKKOS_INLINE_FUNCTION so they can run on GPUs.

3. put all code in one header file (this allows it to be called on a GPU without using Relocatable Device Code)

2 R3D Transfer of Element Scalar

First, recall the linear system described in the remap document:

$$\sum_J \int_\Omega \varphi_I \varphi_J \, \mathrm{d}V g_J = \sum_K \int_\Omega \varphi_I \phi_K \, \mathrm{d}V p_K \tag{3}$$

Where φ are target basis functions and ϕ are donor basis functions, and Ω is the cavity region, which is assumed to remain constant. In the piecewise constant case, both φ and ϕ are one in their element and zero elsewhere. Since target elements do not overlap each other, we have that:

$$\int_\Omega \varphi_I \varphi_J \, \mathrm{d}V = \begin{cases} V(\Omega_I) & I = J \\ 0 & I \neq J \end{cases} \tag{4}$$

Where $V(\Omega_I)$ is the volume of target element I. Target and donor elements do overlap, so on the right hand side we have:

$$\int_\Omega \varphi_I \phi_K \, \mathrm{d}V = V(\Omega_I \cap \Omega_K) \tag{5}$$

This turns Equation 3 into simply:

$$V(\Omega_I) g_I = \sum_K (V(\Omega_I \cap \Omega_K) p_K) \tag{6}$$

Equation 6 represents the scheme now implemented in Omega_h version 2.2.0. It will also operate on vector and tensor fields, by treating each component of the vector/tensor as a scalar field and transferring that.

3 Changing Cavity Boundaries

Edge collapses on curved boundaries (which are implemented and are indispensable) have the potential to create target and donor cavities with different regions. This breaks the original description of conservation which assumes they have the same region Ω:

$$\int_{\Omega} g \, \mathrm{d}V = \int_{\Omega} p \, \mathrm{d}V \tag{7}$$

I think what we would want is that the integral over the target cavity equals the integral over the donor cavity:

$$\int_{\Omega_T} g \, \mathrm{d}V = \int_{\Omega_D} p \, \mathrm{d}V \tag{8}$$

Back in Equation 3, the weighting function w was discretized over the target cavity. It now becomes unclear how to discretize the weighting function. In 3D there exists a case where neither cavity is a strict superset of the other.

Note that the previous Omega_h scheme for element density, although completely diffusive, does keep conservation when the boundary changed.

3.1 Correction for Scalars in Superset Case

Equation 6 suggests a way to deal with this in that it shows we are essentially dealing with sums of intersection volumes. I implemented a correction in case there is a portion of the target element I which is not intersected by donor elements, in which case we only divide by the portion that *does* have intersections:

$$\left(\sum_K V(\Omega_I \cap \Omega_K) \right) g_I = \sum_K (V(\Omega_I \cap \Omega_K) p_K) \tag{9}$$

Setting $p = 1$ and $g = 1$ in Equation 6 shows this is equivalent if the boundary doesn't change. I can think of going one step further and accounting for portions of a donor element which don't intersect the target mesh, which gives:

$$\left(\sum_K V(\Omega_I \cap \Omega_K) \right) g_I = \sum_K \left(V(\Omega_I \cap \Omega_K) \frac{p_K V(\Omega_K)}{\sum_J V(\Omega_J \cap \Omega_K)} \right) \tag{10}$$

Where J, like I, iterates over the target cavity. I have not yet implemented this second correction factor. Also, this only works for piecewise constant element density and the generalization to the nodal velocity case is unclear.

4 Issues with Nodal Cavity Conservation

Even without cavity boundaries changing, there are some more important difficulties for the transfer of the nodal velocity field in such a way that momentum is conserved.

If one treats the cavity as an isolated mesh then the values assigned to cavity boundary nodes could break momentum conservation in elements outside of but adjacent to the cavity.

The current proposed solution is to consider two levels of elements, the first layer being the original cavity, and to impose a condition that the nodes at the two-layer boundary don't change value. If said nodes are removed from the explicit degrees of freedom, we get the same linear system as before except the right hand side has additional terms enforcing momentum conservation of the second-layer elements. If so then the existence and uniqueness of a solution should not be affected. The above claims need to be double-checked before proceeding.

Assuming these claims are true, however, it still means we would be proceeding with two-layer cavities. This implies a number of technical complications for Omega_h and considerable coding effort, but so far no open unanswered questions. In particular, independent set selection would need to proceed with two-layer ghosting, the conflict graph would be a 4th order adjacency instead of the current 2nd order adjacency, and a ghost layer must be re-formed after independent set selection. That last item implies more trouble having to do with ordering and maintaining the relationship between the two meshes, but hopefully this can be sorted out. Adding this two-layer support will be my next work item in the area of solution transfer.

Note that although two layers are required for coarsening and swapping due to the lack of cavity-interior nodes, we could technically use a single layer for refinement and force all momentum conservation to be satisfied by the single central node. However, if this causes the central node value to spike then we may consider two layers for refinement as well.