Difference Imaging in Wide Field Surveys

Ian Bond

Massey University, Albany

2012 Microlensing Meeting, Pasadena

Outline

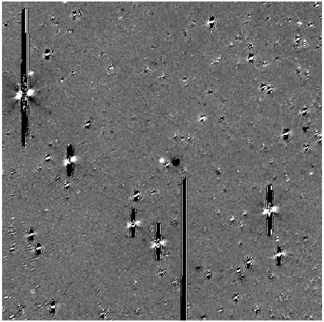
1. What is difference imaging and why do we want to use it?
2. How to implement a spatially varying numerical kernel
3. How to implement difference imaging on a GPU
4. Large Images

Outline

1. What is difference imaging and why do we want to use it?
2. How to implement a spatially varying numerical kernel
3. How to implement difference imaging on a GPU
4. Large Images

What is difference imaging?

- method to detect transient events (including microlensing)
- method to get quality time series photometry of transient events (and variable stars)



MOA implementation

- MOA camera: $10 \times 2048 \times 4096$ CCD mosaic
- Exposure time: 60 secs (for Bulge fields) – generate difference images for all frames in an exposure
  – get positions of possible transients
  – generate "quicklook" photometry for all transients in the list
- All done in real time, i.e before the next exposure finishes
- This approached has worked well for MOA

New Generation Wide Field Surveys

Can we do the same for new generation surveys

- KMT
  - Camera: $4 \times 9\mathrm{K} \times 9\mathrm{K}$
- LSST
  - Camera: $189 \times 4\mathrm{K} \times 4\mathrm{K}$

Important issues

- Can difference imaging be done fast enough for real-time?
- Can it work on large images

Principles of Difference Imaging

- Given two images, r and i, of the same field taken at different times, one constructs a difference image by

$$d(x, y) = i(x, y) - (r * k)(x, y) - b(x, y)$$

- Solve for the “convolution kernel”, k, and the differential background, b
- Both k and b are spatially varying
  – major issue for large images!

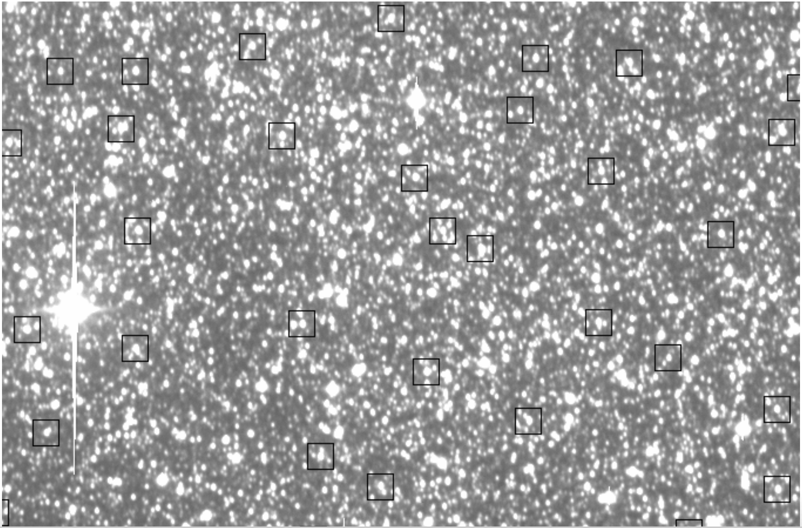
Classic Method

- Model kernel as a linear combination of smooth analytic basis functions (Alard &Lupton 1998)
- Model spatial variation of kernel by 2D polynomial (applied to each coefficient)
- Model background as 2D polynomial

Numerical trick by Alard (1999)

- Kernel is constant on over its own size
- Select a number of “stamps” across the image
- Build zero order matrices for each stamp
- Expand into linear system for the whole image by combining zero order linear systems for each stamp with the basis functions for the spatial variations

Stamps



Classic Method

- Size of the linear system:
  $N = 1 + (N_{\mathrm{k}} - 1)N_{\mathrm{s}} + N_{\mathrm{b}}$
- Typically
  $N_{\mathrm{k}} = 49$ (kernel basis functions)
  $N_{\mathrm{s}} = 6$ (functions to model spatial variations in k
  $N_{\mathrm{b}} = 6$ (functions to model background)
  $N = 295$
- All takes <10 seconds on a modern PC for a single 2K $\times$ 4K image
  – Fast enough? Maybe.

Numerical Kernel

- Treat kernel as a numerical table and solve for each entry (Bramich 2008)
- Used extensively for follow-up observations (small telescopes, small CCDs)
  – weird kernel shapes, not necessary to interpolate (Albrow 2009)
- More computationally expensive than the classic method
- Can we use the numerical kernel for wide field surveys?
  – Need to allow for for spatially varying kernel
  – Need a reasonable computation time

Outline

Spatially varying numerical kernel

- A spatially varying numerical kernel can be implemented using the principles of the classic method
- The basis function set is just a grid of delta functions.
- Modify slightly to apply flux conservation rule:

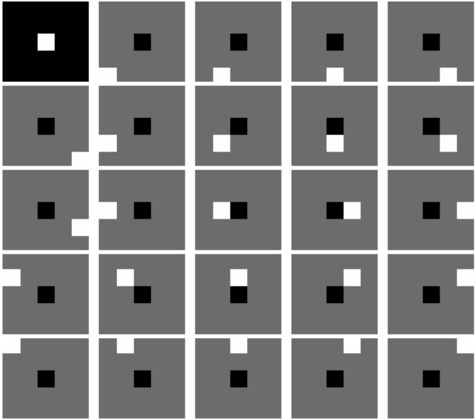
$$k(x, y) = c_0\delta(x, y) + \sum_n c_i \left[\delta(x - j, y - i) - \delta(x, y)\right]$$

  The c_i are spatially dependent (modelled using 2d polynomial set), but the photometric scaling is always given by c_0
- Also include background

$$b(x, y) = \sum_n c_n x^i y^j$$

Spatially varying numerical kernel

A numerical 5×5 kernel basis set



Linear system becomes largel

- For a 17×17 numerical kernel, we need to build a 289×289 zero order matrix for each stamp
- Degree 2 polynomial for spatial variations in the kernel (6 basis functions)
- Degree 2 polynomial for background (6 basis functions)
- Total size of linear system $N = 1 + 288 \times 6 + 6 = 1735$

Can we speed things up?

Outline

NVIDIA GeForce GTX 480

Programming a GPU

GPU architecture supports Single Instruction Multiple Thread programming model

- executes instruction for all threads in the warp, before moving onto next instruction
- divergence occurs when threads in a warp follow different control flows. Sequential passes are then needed which can affect performance
- but SIMT approach is ideal for most image processing operations

Examples

- Zero order matrices

$$Q_{sij} = \sum_{(x,y)\epsilon S_s} [r(x - x_i, y - y_i) r(x - x_j, y - y_j) - r(x,y) r(x,y)]$$

each thread works on its own s, i, j

- Expanding zero order matrices

$$M_{ij} = \sum_{\text{stamps}} F(s, s_i) F(s, s_j) Q(s, k_i, k_j)$$

one thread per i and j

- Convolution

$$(r * k)(x, y) = \sum_{(x_k, y_k)} r(x_k, y_k) k(t, x_i, y_i)$$

one thread per pixel x, y (t is “tile” index)

Performance

2 K $\times$ 4 K image
17 $\times$ 17 spatially variable numerical kernel

	CPU	GPU
Build zero order system	30 secs	2 secs
Expand over stamps	6 secs	72 ms
Solve linear system	10 secs	10 secs
Build difference image	5 secs	91 ms

Outline

Dealing with large images

Work in progress

- For large images, 2nd order polynomials may not be be able to model high frequency spatial variations in the kernel and background
- One approach is to divide image into tiles and solve separately for each.
- May have issues with continuity across the tile boundaries
- Can we force continuity conditions on the polynomial models?

Hermite polynomials

- Form a set of 16 2D polynomials, $h_{ij}(u, v) = h_i(u)h_j(v)$ from the 1D Hermite blending functions
  $h_1(u) = 1 - 3u^2 + 2u^3$
  $h_2(u) = u - 2u^2 + u^3$
  $h_3(u) = 3u^2 - 2u^3$
  $h_4(u) = -u^2 + u^3$
  where $u = (x - x_i)/w$, $v = (y - y_i)/w$ for tile i
- each tile has its own 16 2D polynomial set
- but coefficients are shared at each node of the tile

Hermite polynomials

- for any arbitrary set of coefficients, the resulting surface satisfies at the tile boundaries:
  C_0: functions continuous
  C_1: 1st derivatives contuous
- total of $4(N_x + 1)(N_y + 1)$ coefficients
- could use this basis set to model spatial variations in the kernel, and background
- end up solving a single large linear system

Tiling