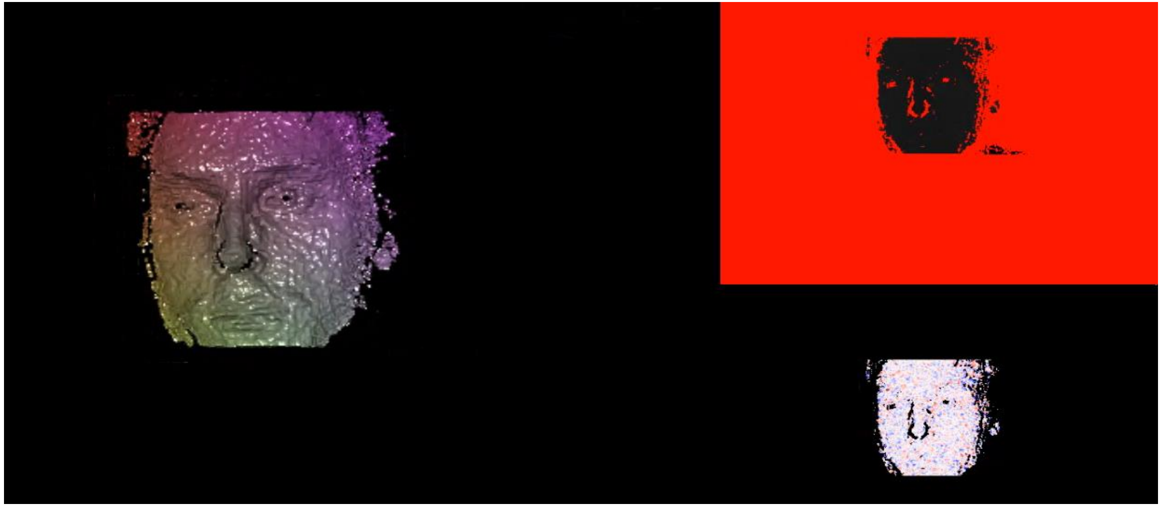
Go With The (Optical) Flow

Phil Noonan
Jan, 2018

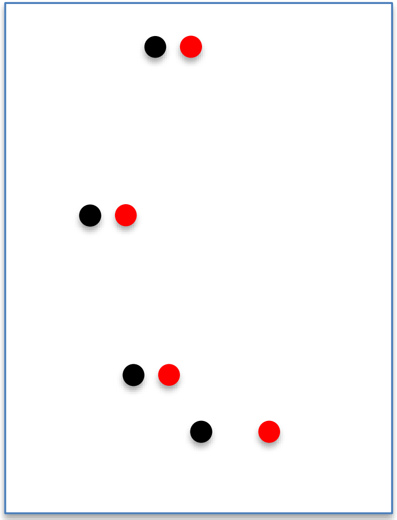
Overview

- KinectFusion
- ICP
- Sparse Features
- Optical Flow
- Fast Optical Flow
- Dense Optical Flow
- Fast HD Dense Optical Flow
- openGL implementation
- Examples
- Future Work

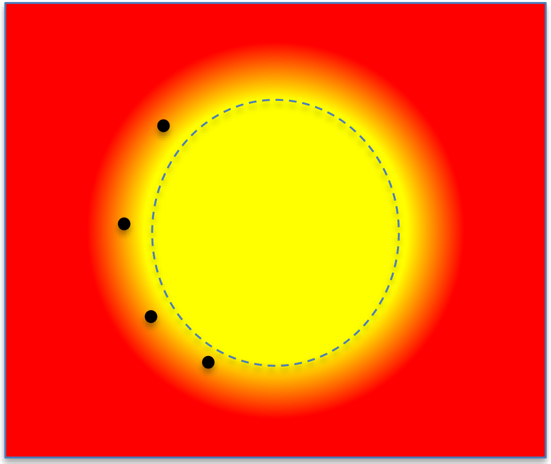
KinectFusion



ICP



Depth to Depth

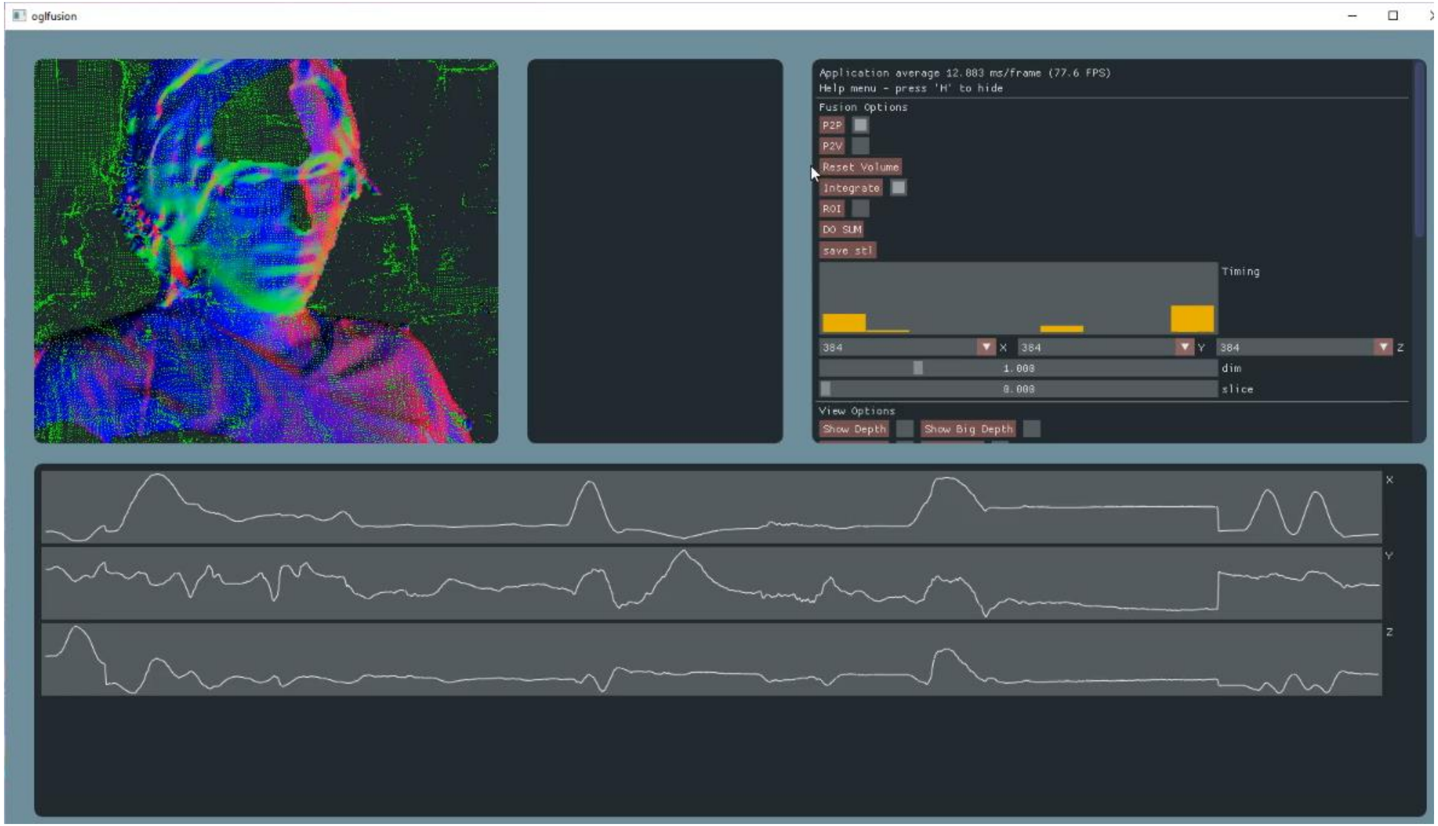


Depth to Volume

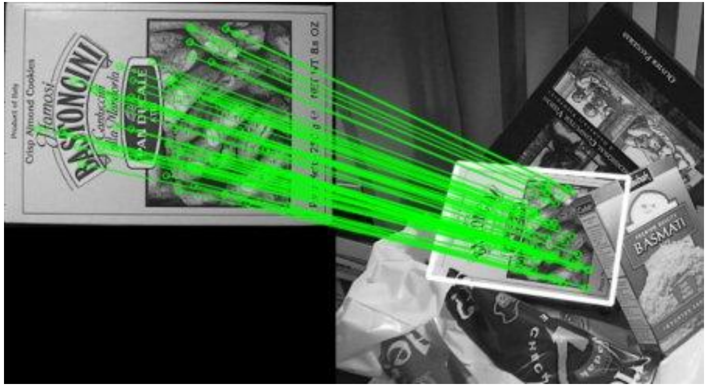
Flowverlay

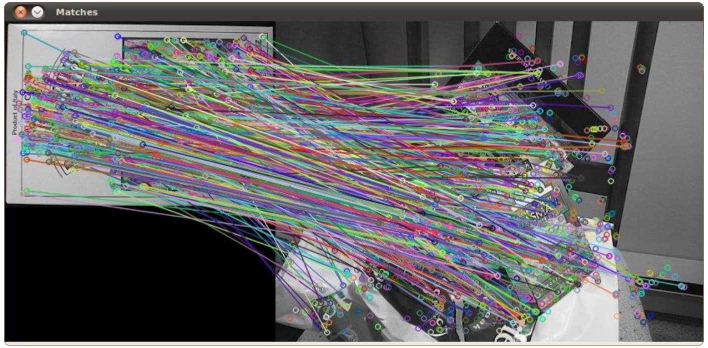
Green points – pixels shifted using flow

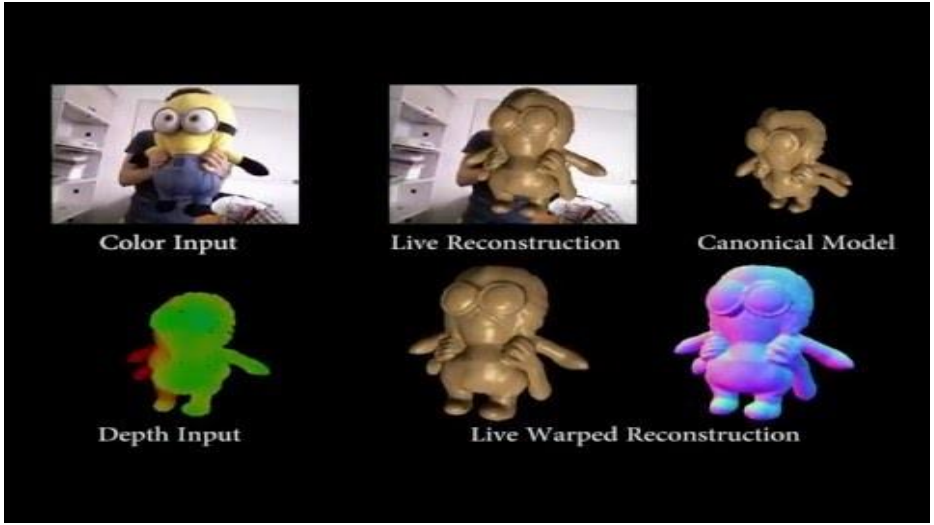
Blue/Green/Red surfaces – normals



Sparse Features







https://www.youtube.com/watch?v=lk_yX-O_Y5c

Optical Flow

Lukas-Kanade Algorithm

Minimise the sum of squared error between two images, template T and image I warped back to the coordinate frame of T

$$\sum_{\mathbf{x}} \left[I\left(\mathbf{W}(\mathbf{x};\mathbf{p})\right) - T(\mathbf{x}) \right]^2 \qquad \mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} x + p_1 \\ y + p_2 \end{pmatrix}$$

Non-linear, pixel values are un-related to pixel coordinates
LK assumes initial flow is known and solves for incremental updates

$$\sum_{\mathbf{x}} \left[I\left(\mathbf{W}(\mathbf{x};\mathbf{p} + \Delta\mathbf{p})\right) - T(\mathbf{x}) \right]^2$$

Optical Flow

$$\sum_{\mathbf{x}} \left[I\big(\mathbf{W}(\mathbf{x};\mathbf{p})\big) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2$$

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$$

Image gradients

If…

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \left(W_x(\mathbf{x};\mathbf{p}), W_y(\mathbf{x};\mathbf{p}) \right)^T$$

then…

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}$$

Jacobian

For 2D affine case

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}$$

Optical Flow

The partial derivative of the first expression with respect to $\Delta\mathbf{p}$ is

$$2\sum_{\mathbf{x}}\left[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[I(\mathbf{W}(\mathbf{x};\mathbf{p}))+\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}-T(\mathbf{x})\right]$$

Setting this = 0 and solving gives closed form solution

$$\Delta\mathbf{p}=H^{-1}\sum_{\mathbf{x}}\left[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}[T(\mathbf{x})-I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$$

Where H is Gauss-Newton approximation to the Hessian matrix

$$H=\sum_{\mathbf{x}}\left[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]$$

Optical Flow

1. Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
2. Compute error $T(\mathrm{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
3. Warp the gradient ∇I with $\mathbf{W}(\mathbf{x};\mathbf{p})$
4. Evaluate Jacobian
5. Compute steepest descent
6. Compute Hessian
7. Compute $\Delta\mathbf{p}$

Fast Optical Flow

Reverse the roles of I and T and minimise

$$\sum_{\mathbf{x}} [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,]^2$$

First Order Taylor expansion gives

$$\sum_{\mathbf{x}} \left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2$$

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]$$

Assuming $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp

$$H = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Fast Optical Flow

$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

$$H = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Inverse composition

Fast Optical Flow

Precompute

1. Evaluate the gradient ∇T of the template $T(\mathbf{x})$
2. Evaluate the Jacobian
3. Compute steepest descent images
4. Compute Hessian

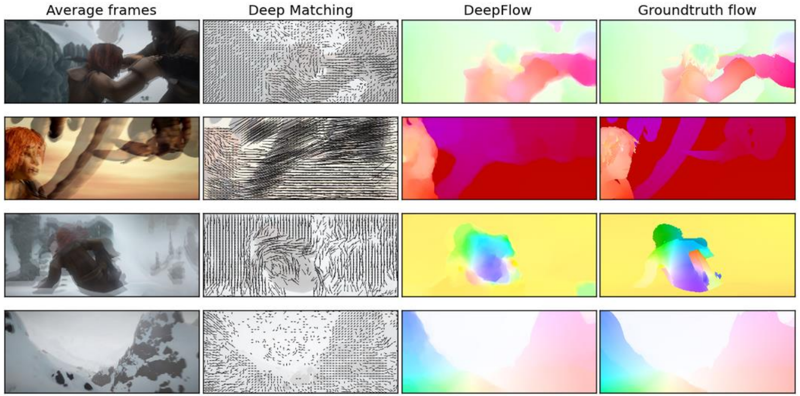
Iterate

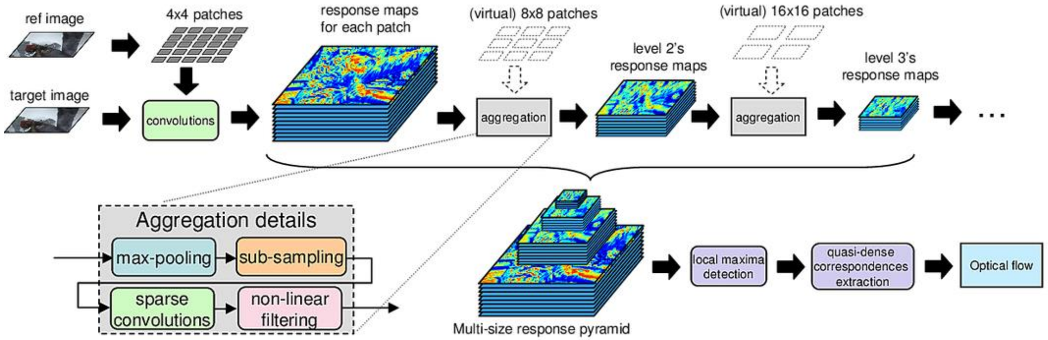
1. Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
2. Compute error $T(\mathrm{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
3. Compute $\Delta\mathbf{p}$
4. Update warp

Dense Optical Flow

DeepFlow

Philippe Weinzaepfel Jerome Revaud Zaid Harchaoui Cordelia Schmid





Fast Dense Optical Flow 1

https://arxiv.org/pdf/1603.03590.pdf ECCV 2016 Kroeger et al

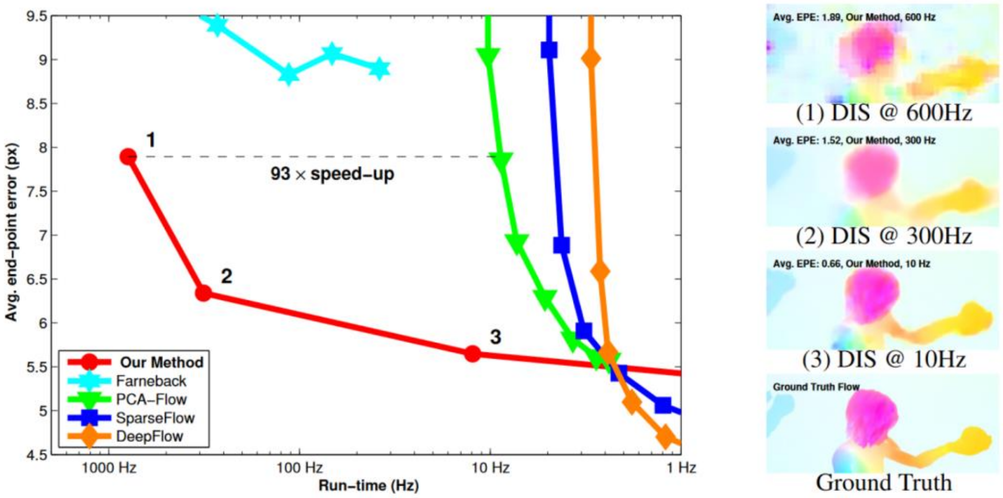


Figure 1: Our DIS method runs at 10Hz up to 600Hz on a single core CPU for an average end-point pixel error smaller or similar to top optical flow methods at comparable speed. This plot excludes preprocessing time for *all* methods. Details in § 3.1,3.3.

Fast Dense Optical Flow 2

Dense Inverse Search optical Flow

Create image pyramids
Process each level from coarse to fine
Create grid of patches over the image domain
Patches uniformly overlap

Iterate
    If coarsest level, initialize flow = 0, else initialise flow from level below
    Inverse Search
    Densification of overlapping patches

Fast HD Dense Optical Flow

DisOptFlow is implemented in OpenCV Contrib
~500 ms per 1920x1080 image

Can we do better on GPU?

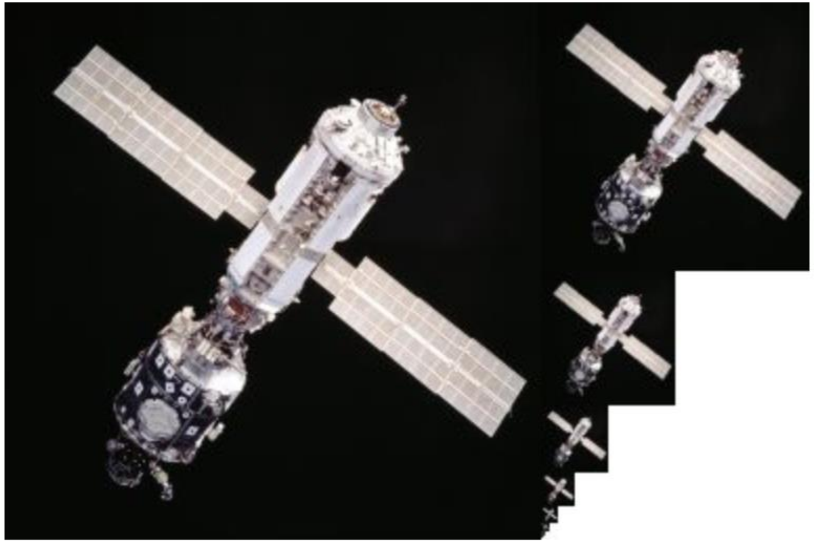
openGL Mipmap

Image pyramid creation requires successive downsampling and interpolation at each level
i.e. multiple texel reads per pixel at each level

```
cv::resize(Src, Dest[i], Dest[i].size(), 0.0, 0.0, cv::INTER_AREA);
```

Or…

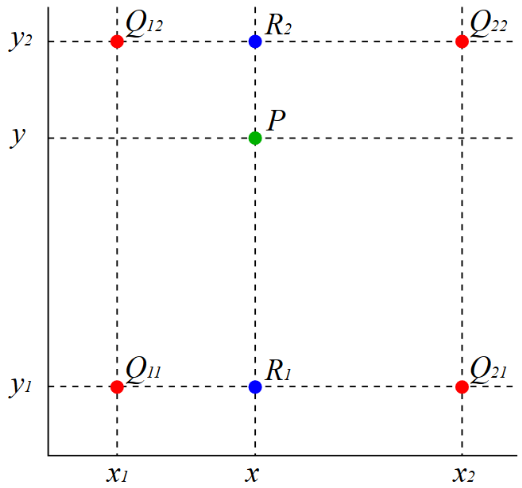
```
glGenerateMipmap(GL_TEXTURE_2D);
```



Interpolation

```
#define INIT_BILINEAR_WEIGHTS(Ux, Uy)
i_I1 = min(max(i + Uy + bsz, i_lower_limit), i_upper_limit);
j_I1 = min(max(j + Ux + bsz, j_lower_limit), j_upper_limit);
w11 = (i_I1 - floor(i_I1)) * (j_I1 - floor(j_I1));
w10 = (i_I1 - floor(i_I1)) * (floor(j_I1) + 1 - j_I1);
w01 = (floor(i_I1) + 1 - i_I1) * (j_I1 - floor(j_I1));
w00 = (floor(i_I1) + 1 - i_I1) * (floor(j_I1) + 1 - j_I1);


 diff = w00 * I1_ptr[i * I1_stride + j] +
         w01 * I1_ptr[i * I1_stride + j + 1] +
        w10 * I1_ptr[(i + 1) * I1_stride + j] +
        w11 * I1_ptr[(i + 1) * I1_stride + j + 1] -
              I0_ptr[i * I0_stride + j];
```



openGL Linear Interpolation

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST);
```

```
I0_val = textureLod(tex_I0, vec2(xCoord, yCoord), level).xyz;
I1_val = textureLod(tex_I1, vec2(xCoord + u, yCoord + v), level).xyz;
```

Read from a texture, manually specifying level of detail

At this coordinate (normalized from 0.0f - 1.0f)

At this mipmap integer level

openGL Shared Memory

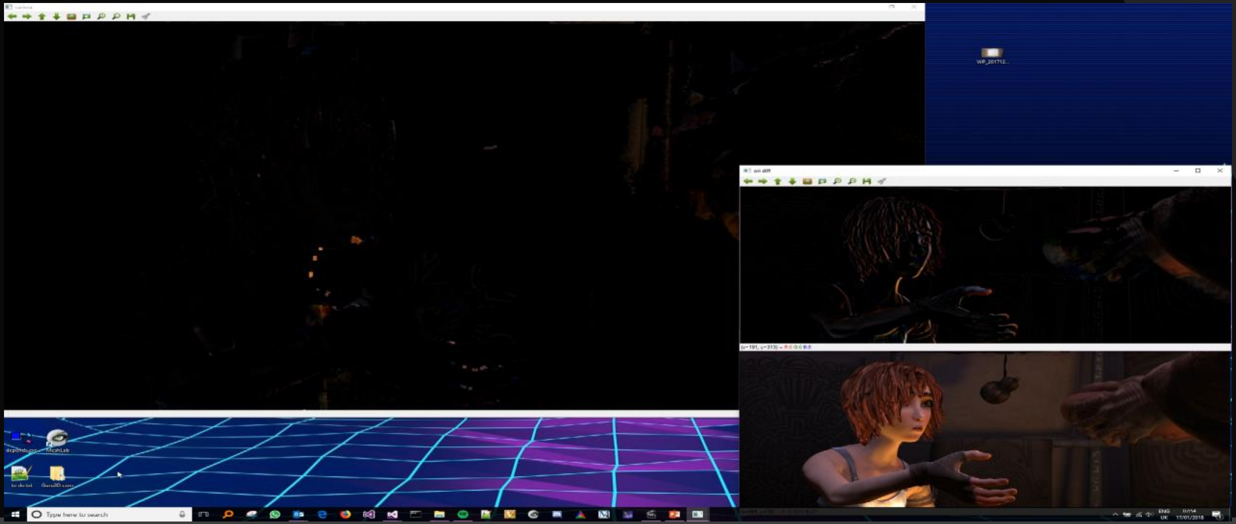
```
shared float I0_data[20][20];
shared float I0x_grad_data[20][20];
shared float I0y_grad_data[20][20];
```

Each GPU thread is responsible for 1 patch
Workgroup consists of 4 x 4 threads
Each thread reads in 5 x 5 pixels and loads to shared memory

Reduces frequency of global memory access
Used in gradient operator, patch generation, and inverse search

Synthetic test data

1 Girl 1 Cup



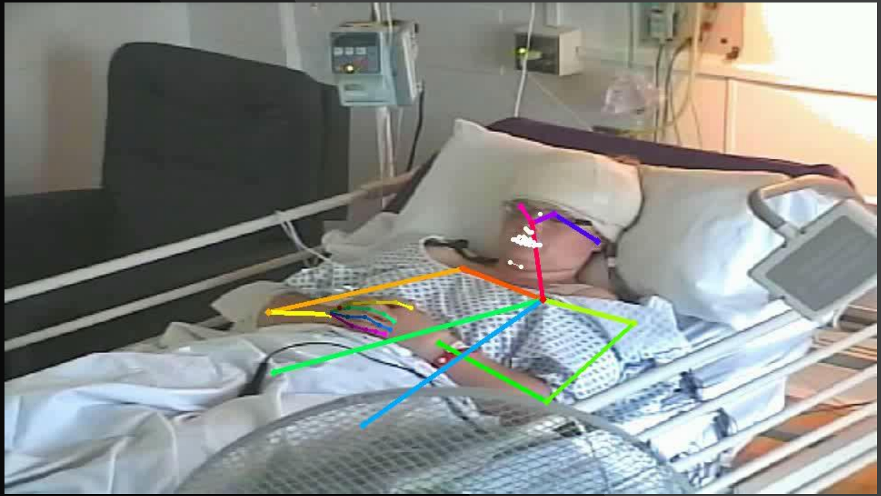
Difference of frames

Original video

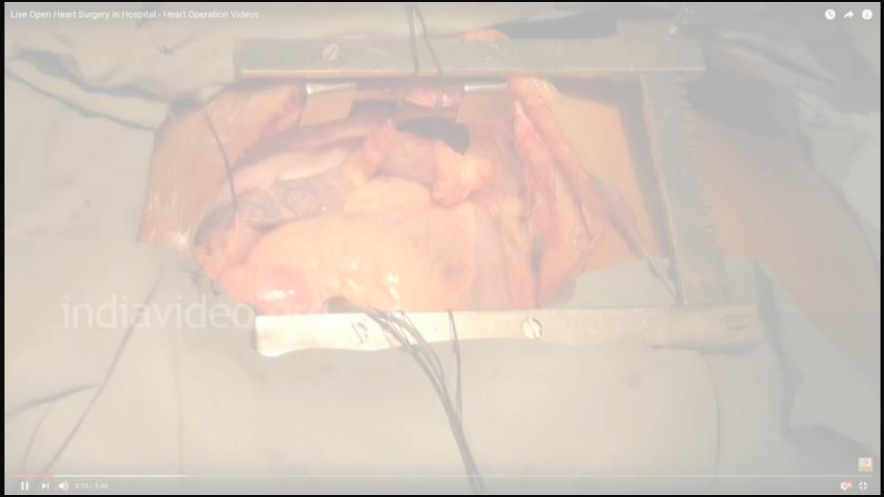
Tonic Clonic Seizures

Tonic Clonic Seizures

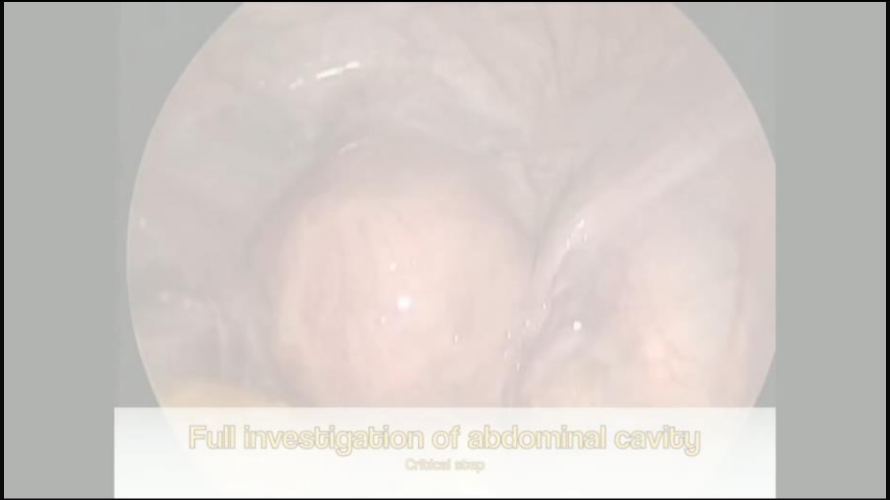
openPose



Open Heart


Live Open Heart Surgery in Hospital - Heart Operation Videos
indiavideo

Laproscopy



Laproscopy with Tool

Future Improvements

- Initial Flow Estimate Improvements
- CoEstimate Flow and Depth
- Intel realsense – 90 Hz 720p Depth
- Vulkan – mobile devices + faster on PC
- Depth From Stereo

Refs

Baker and Matthews, ‘Lucas-Kanade 20 Years On: A Unifying Framework’

https://www.ri.cmu.edu/pub_files/pub3/baker_simon_2002_3/baker_simon_2002_3.pdf

Kroeger et al, ‘Fast Optical Flow using Dense Inverse Search’

https://arxiv.org/pdf/1603.03590.pdf

Revaud et al, ‘DeepMatching: Hierarchical Deformable Dense Matching’

https://thoth.inrialpes.fr/src/deepflow/

Innmann et al, ‘VolumeDeform: Real-time Volumetric Non-rigid Reconstruction’

https://graphics.stanford.edu/~niessner/papers/2016/5volumeDeform/innmann2016deform.pdf

Thanks

More Maths

compositional algorithm is equivalent to it also. The proof of equivalence here takes a very different form to the proof in Section 3.1.5. The first step is to note that the summations in Eqs. (12) and (31) are discrete approximations to integrals. Equation (12) is the discrete version of:

$$\int_T [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x})\,]^2 \, d\mathbf{x} \tag{37}$$

where the integration is performed over the template T. Setting $\mathbf{y} = \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$, or equivalently $\mathbf{x} = \mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1}$, and changing variables, Eq. (37) becomes:

$$\int_{\mathbf{W}(T)} [I(\mathbf{W}(\mathbf{y};\mathbf{p})) - T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1})]^2 \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}} \right| d\mathbf{y} \tag{38}$$

where the integration is now performed over the image of T under the warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ which we denote: $\mathbf{W}(T) = \{\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \mid \mathbf{x} \in T\}$. Since $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, it follows that:

$$\left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}} \right| = 1 + O(\Delta\mathbf{p}). \tag{39}$$

The integration domain $\mathbf{W}(T)$ is equal to $T = \{\mathbf{W}(\mathbf{x};\mathbf{0}) \mid \mathbf{x} \in T\}$ to a zeroth order approximation also. Since we are ignoring higher order terms in $\Delta\mathbf{p}$, Eq. (38) simplifies to:

$$\int_T [T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2 \, d\mathbf{x}. \tag{40}$$

In making this simplification we have assumed that $T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))$, or equivalently $T(\mathbf{y}) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))$, is $O(\Delta\mathbf{p})$. (This assumption is equivalent to the assumption made in Hager and Belhumeur (1998) that the current estimate of the parameters is approximately correct.) The first order terms in the Jacobian and the area of integration can therefore be ignored. Equation (40) is the continuous version of Eq. (31) except that the term $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is inverted. The estimate of $\Delta\mathbf{p}$ that is computed by the inverse compositional algorithm using Eq. (31) therefore gives an estimate of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ that is the inverse of the incremental warp computed by the compositional algorithm using Eq. (12). Since the inverse compositional algorithm inverts $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ before composing it with $\mathbf{W}(\mathbf{x};\mathbf{p})$ in Step 9, the two algorithms take the same steps to first order in $\Delta\mathbf{p}$.