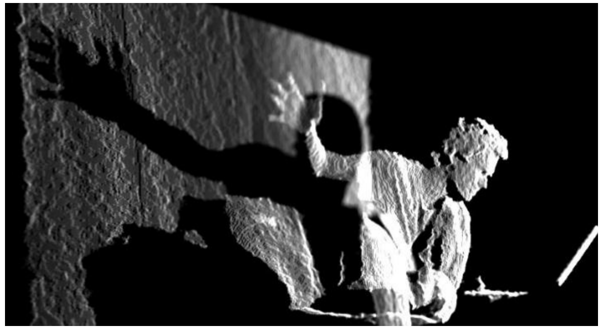
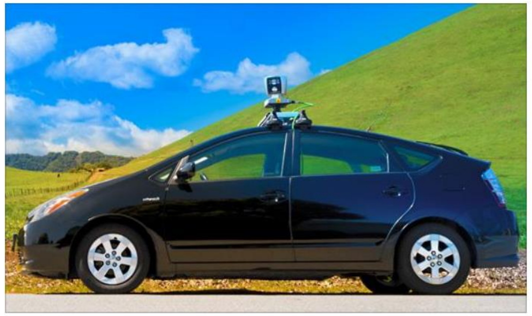
3D Scanning & Motion Capture

Optimization Methods for 3D Reconstruction

Prof. Matthias Nießner

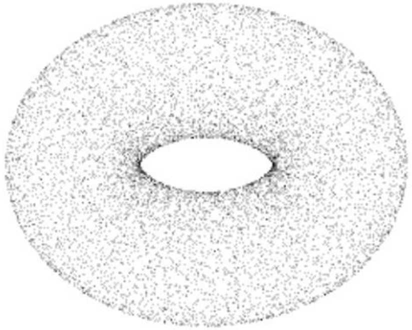
Last Lecture: How to obtain "3D"?



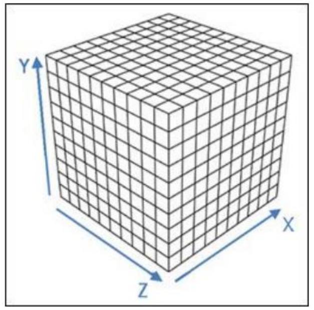


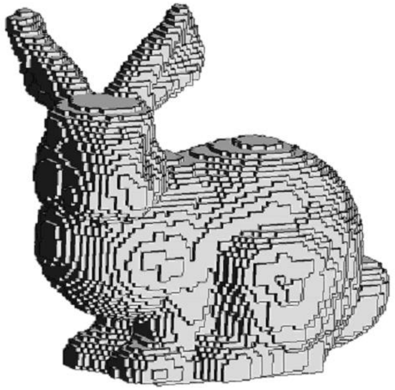
Last Lecture: Surface Representations

- Point Clouds

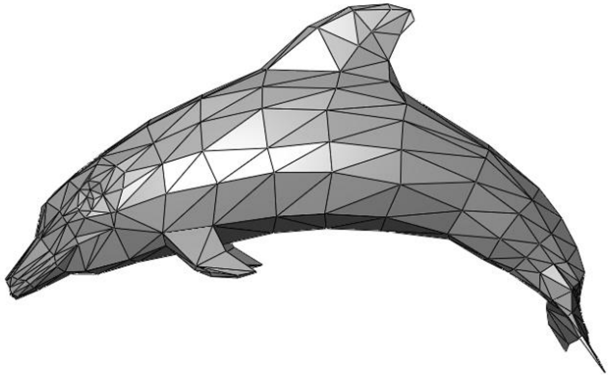


- Voxels

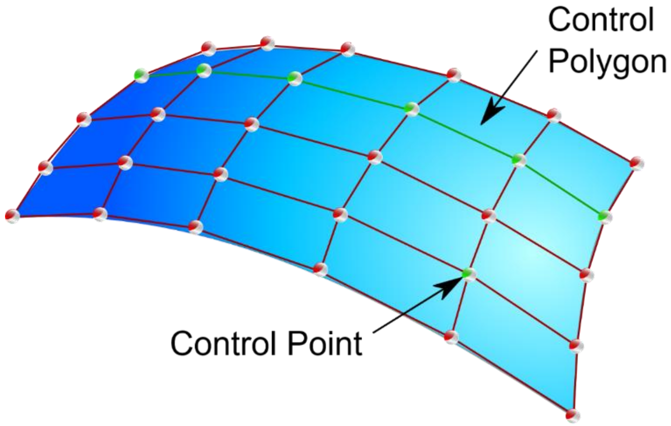




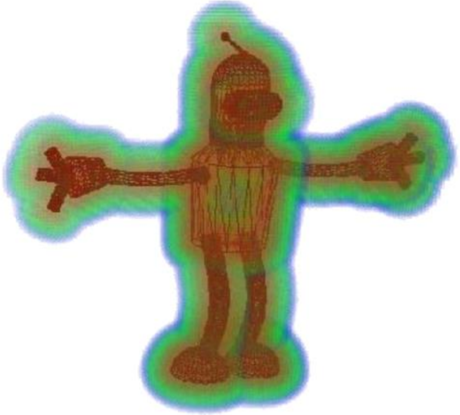
- Polygonal Meshes

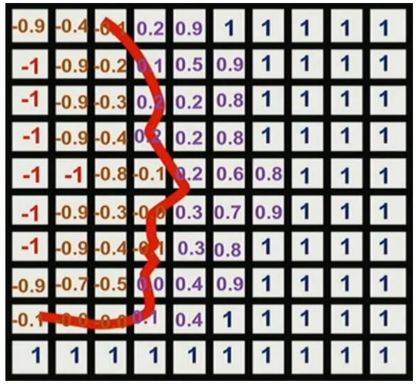


- Parametric Surfaces



- Implicit Surfaces

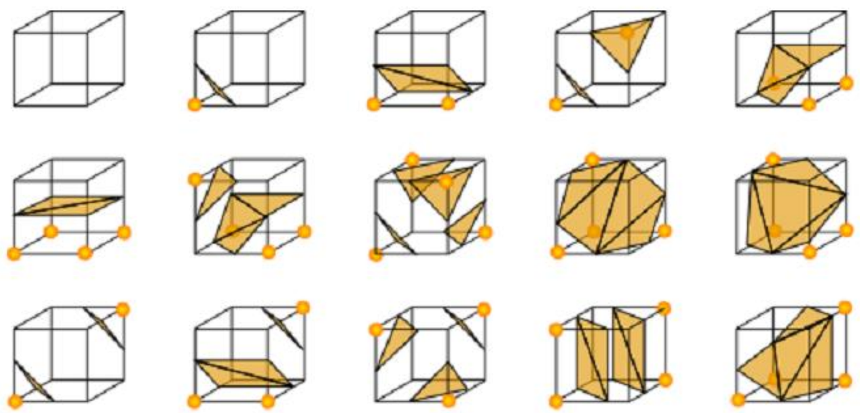




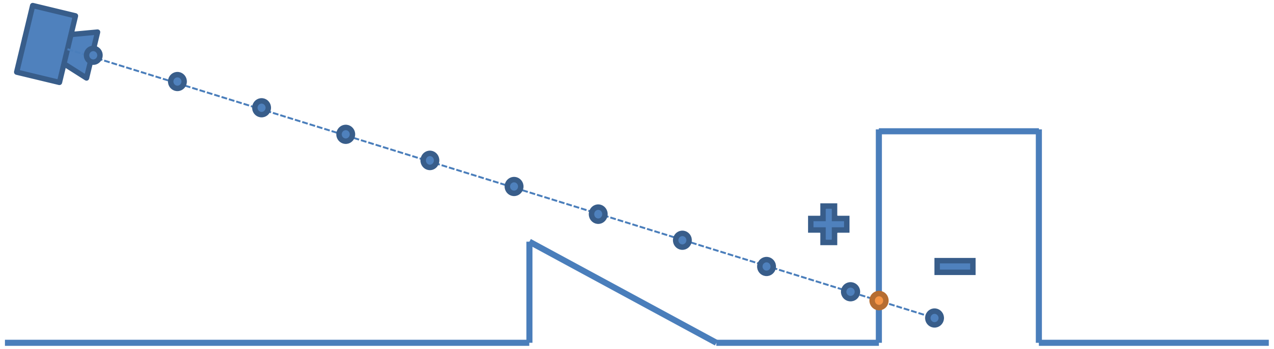
Truncated Signed Distance Field (TSDF)

Last Lecture: Surface Representations

- Important Algorithms
  - Marching Cubes
  - Ray cast

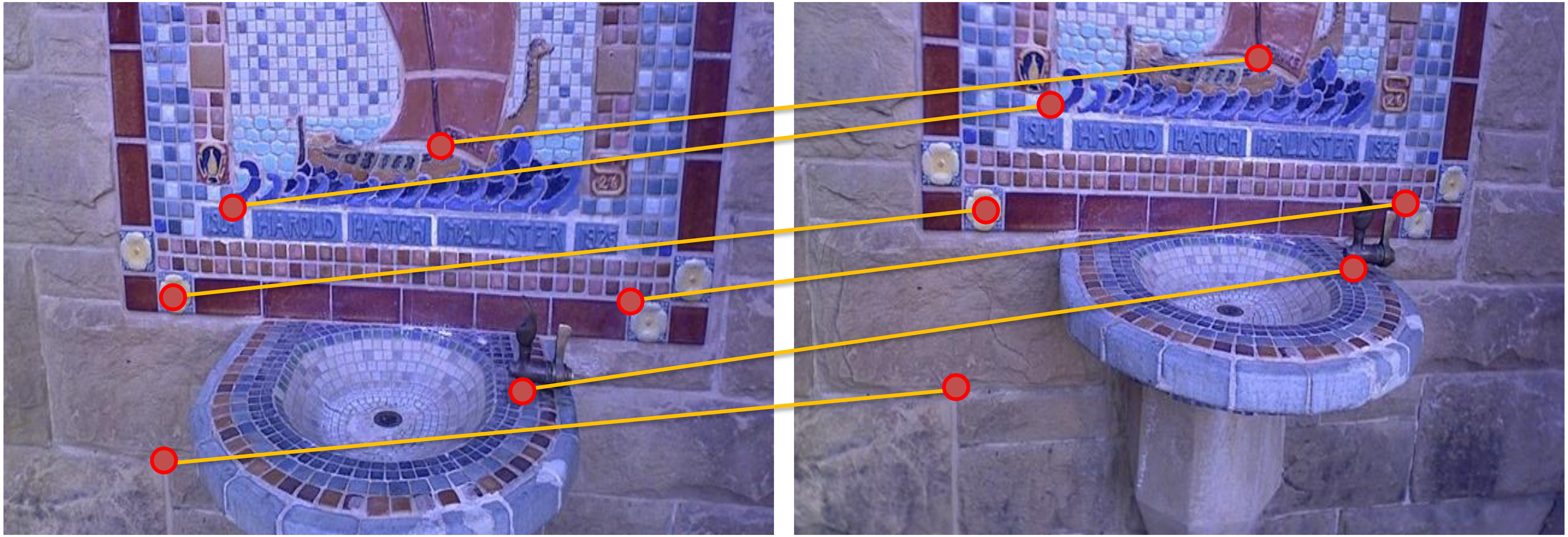


Marching Cubes table

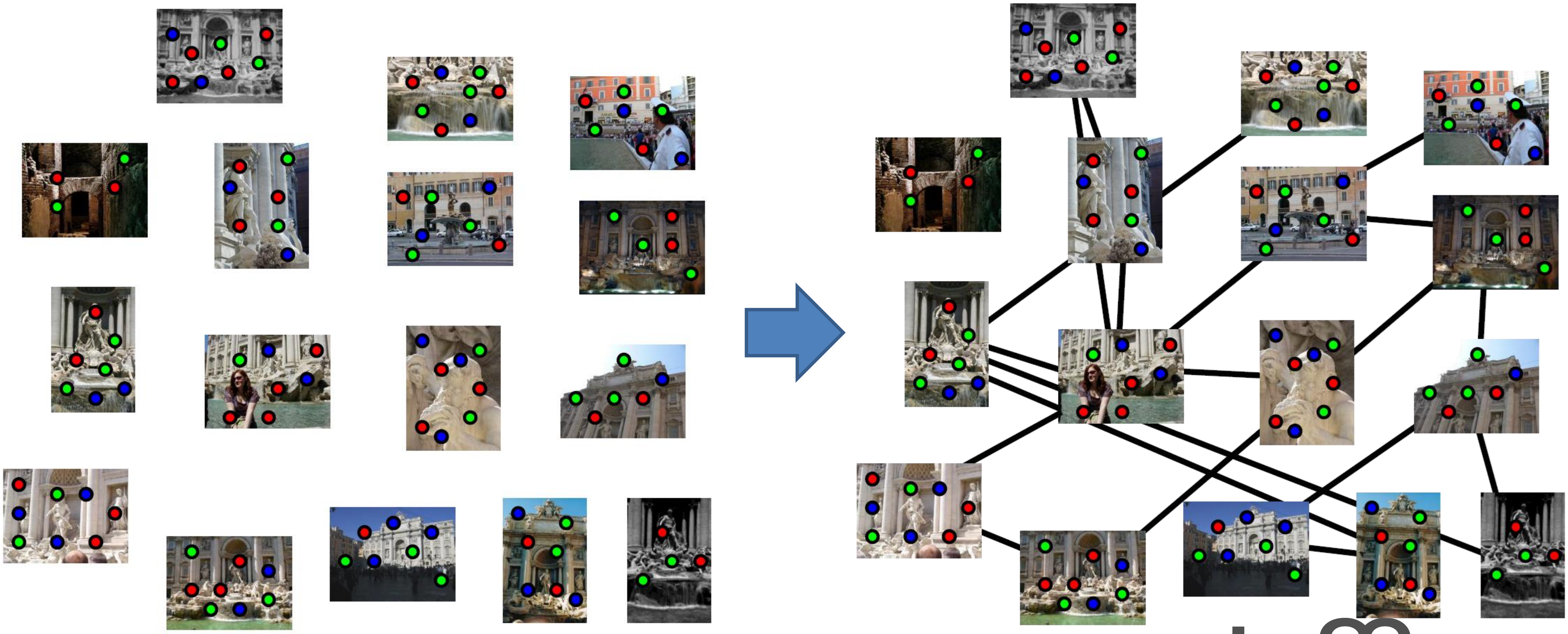


Ray Casting

Last Lecture: Correspondence Finding / Matching



Last Lecture: Correspondence Finding / Matching

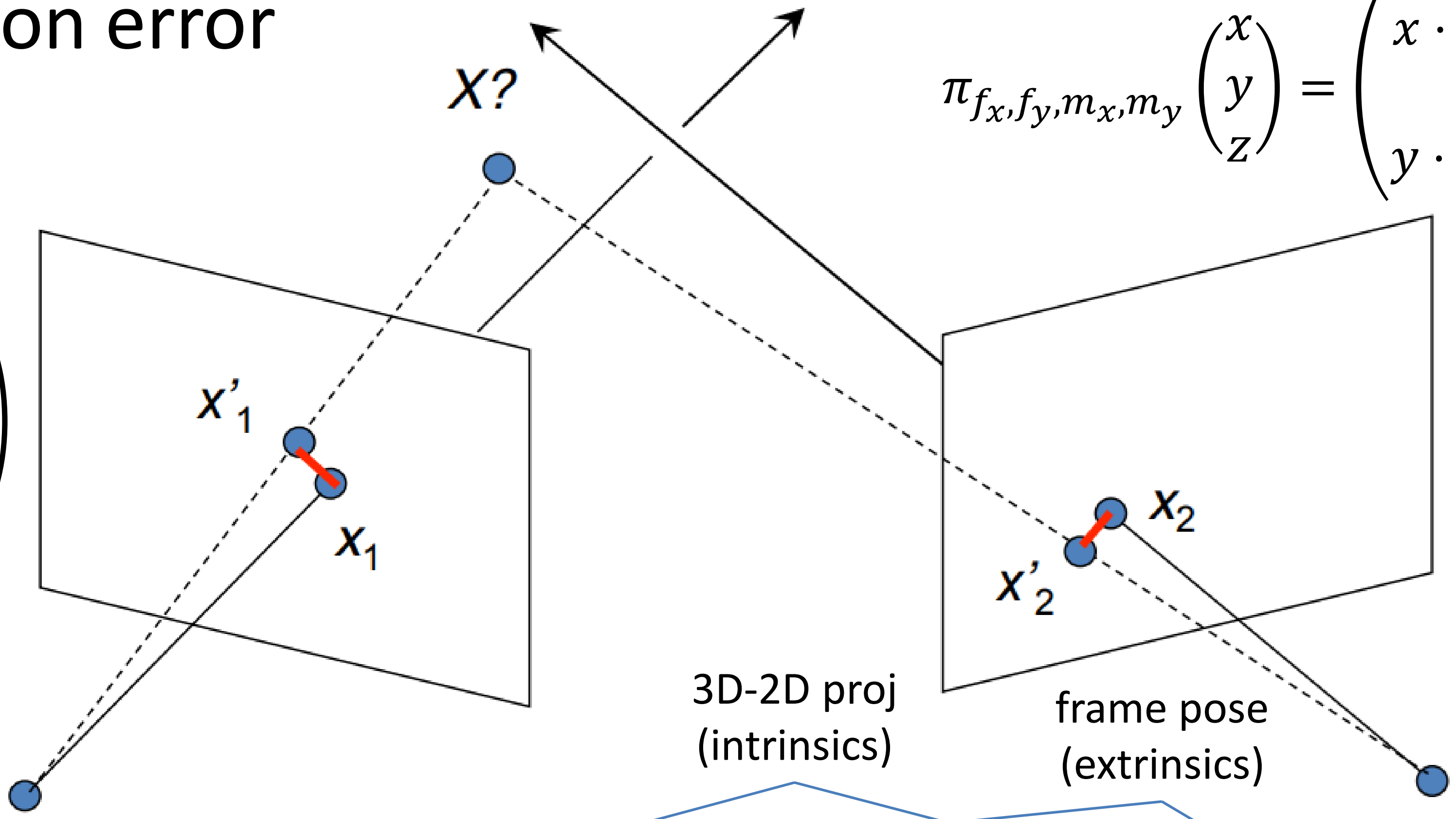


Last Lecture: Bundle Adjustment (SfM)

- Re-projection error

$$T(\alpha, \beta, \gamma, t_x, t_y, t_z) = \begin{pmatrix} R_{00} & R_{01} & R_{02} & t_x \\ R_{10} & R_{11} & R_{12} & t_y \\ R_{20} & R_{21} & R_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\pi_{f_x, f_y, m_x, m_y}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \cdot \frac{f_x}{z} + m_x \\ y \cdot \frac{f_y}{z} + m_y \end{pmatrix} \rightarrow \begin{pmatrix} u \\ v \end{pmatrix}$$



3D-2D proj
(intrinsics)

frame pose
(extrinsics)

$$E(T_{left}, T_{right}, X) = \left\|x_1 - \pi_{left}(T_{left} \cdot X)\right\|_2^2 + \left\|x_2 - \pi_{right}(T_{right} \cdot X)\right\|_2^2$$

Last Lecture: Bundle Adjustment (SfM)

- m images
- n points in 3d

- $E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) =$
  $\sum_{i=1}^{m} \sum_{j=1}^{n} \left\| x_{ij} - \pi_i(T_i \cdot X_j) \right\|_2^2$

over images

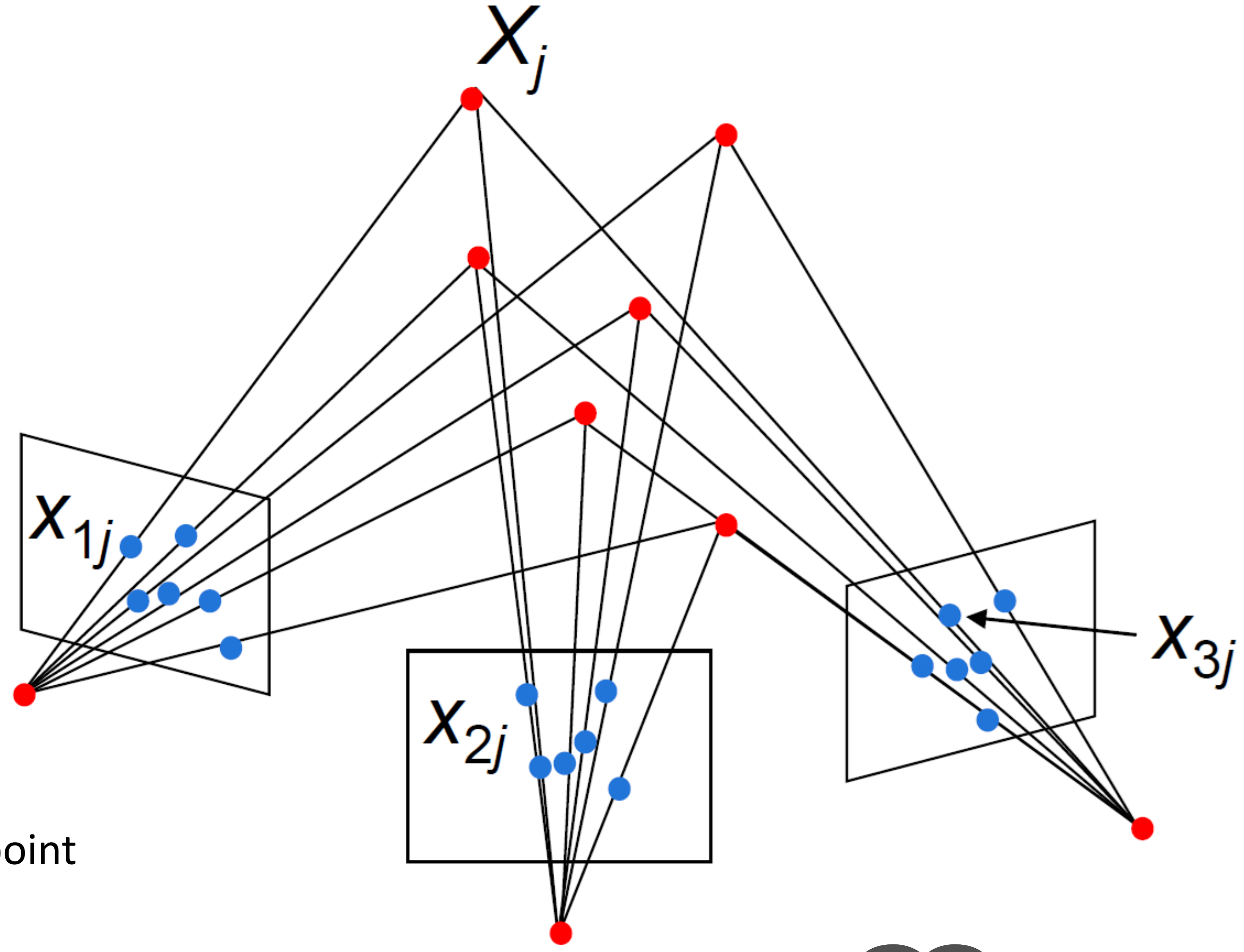
over 3d points

2d keypoint locations

3D-2D proj (intrinsics)

frame pose (extrinsics)

3D point



Last Lecture: RGB-D "Bundling"

$$E_{bundle}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

$$E_{depth}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \left(p_k - T_i^{-1} T_j \pi_d^{-1} \left(D_j \left(\pi_d \left(T_j^{-1} T_i p_k \right) \right) \right) \right) \cdot n_k \right\|_2^2$$

$$E_{color}(T) = \sum_{i,j}^{\#\text{frames}} \sum_{k}^{\#\text{pixels}} \left\| \nabla I \left(\pi_c (p_k) \right) - \nabla I \left(\pi_c \left(T_j^{-1} T_i p_k \right) \right) \right\|_2^2$$

Today: Optimization Methods for 3D Reconstruction

How do we solve these non-linear terms?

- Bundle Adjustment or RGB-D Bundling

$$E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left\| x_{ij} - \pi_i(T_i \cdot X_j) \right\|_2^2$$

$$E_{keypoint}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

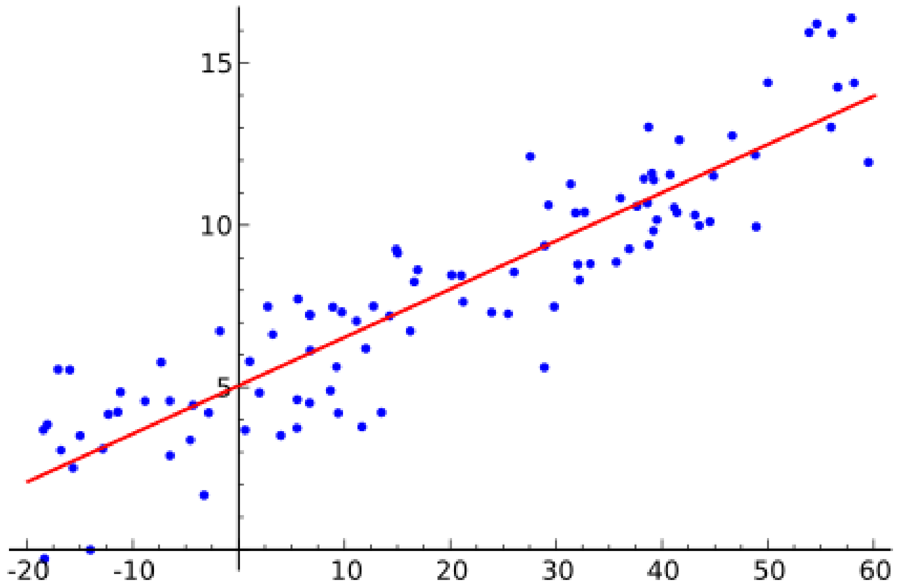
Least Squares

- Find solution that minimizes the sum of squared residuals
  - $f(x) = \sum r_i(x)^2$
  - $f(x) = ||F(x)||_2^2 \, , F(x) = [r_1(x), r_2(x), \ldots, r_n(x)]^T$
  - $x^* = \underset{x}{\operatorname{argmin}} f(x) = \underset{x}{\operatorname{argmin}} ||F(x)||_2^2$
  - $f(x) = \left|\left| \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_n(x) \end{bmatrix} \right|\right|_2^2$

Linear Least Squares

- Linear function: $y = m \cdot x + t$
  - Solve for m, t

- $r_i(m, t) = y_i - (m \cdot x_i + t)$



Linear Least Squares

- $r_i(m,t) = y_i - (m \cdot x_i + t)$
- $x_i = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, y_i = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$
- $A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}, b = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$
- $Ax = b$ (over determined)
- Solve via normal equation $A^T Ax = A^T b$

- $A^T A = \begin{bmatrix} 30 & 10 \\ 28 & 4 \end{bmatrix}$ $A^T b = \begin{bmatrix} 77 \\ 28 \end{bmatrix}$
- Solve: $\begin{bmatrix} 30 & 10 \\ 28 & 4 \end{bmatrix} \begin{bmatrix} m \\ t \end{bmatrix} = \begin{bmatrix} 77 \\ 28 \end{bmatrix}$

-> $\begin{bmatrix} m \\ t \end{bmatrix} = \begin{bmatrix} 3.5 \\ 1.4 \end{bmatrix}$

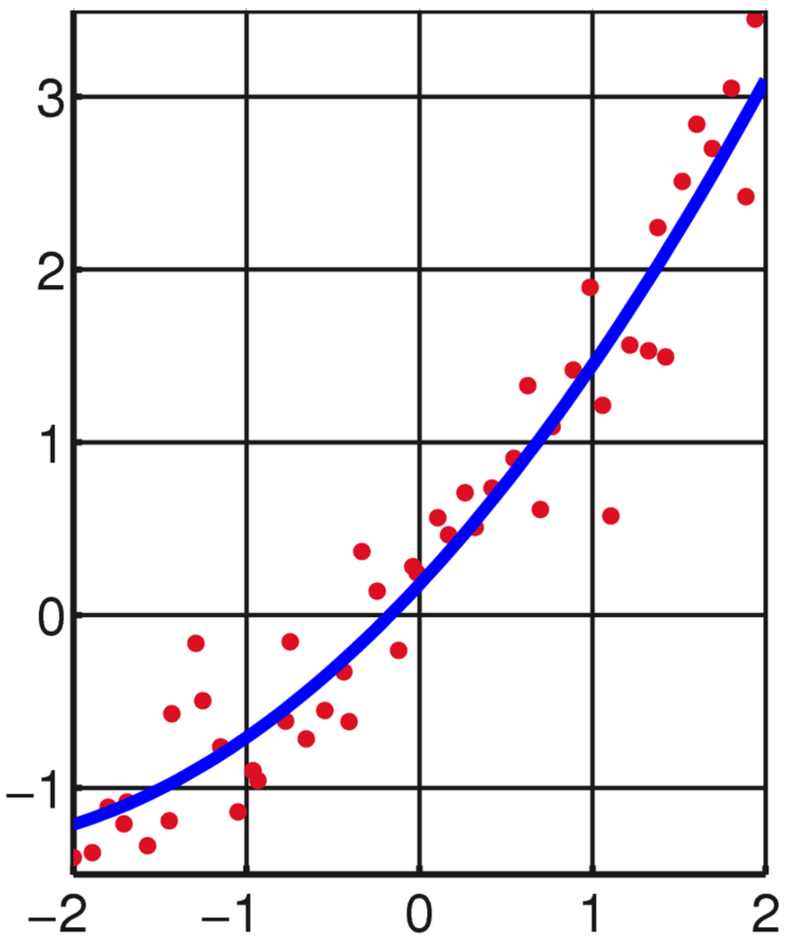
Linear Least Squares

- Quadratic function: $y = ax^2 + bx + c$
  - Solve for a, b, c
  - Linear with respect to a, b, c

- $r_i(a, b, c) = y_i - (ax^2 + bx + c)$

- $A = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \\ 16 & 4 & 1 \end{bmatrix}, b = \begin{bmatrix} 6 \\ 5 \\ 7 \\ 10 \end{bmatrix}$

- $Ax = b$ (over determined)
- Solve via normal equation $A^T A x = A^T b$



Linear Least Squares

- Solve Normal Equation: $A^T A x = A^T b$
  - Compute Matrix Inverse?
  - Gradient descent?
- Linear Solve (iterative):
  - Jacobi Iteration
  - Gauss-Seidel Iteration
  - Conjugate Gradient Descent
- Linear Solve (direct):
  - QR-, LU-Decomposition
  - Cholesky Decomposition
  - SVD
  - …

Hard to write a good solver yourself

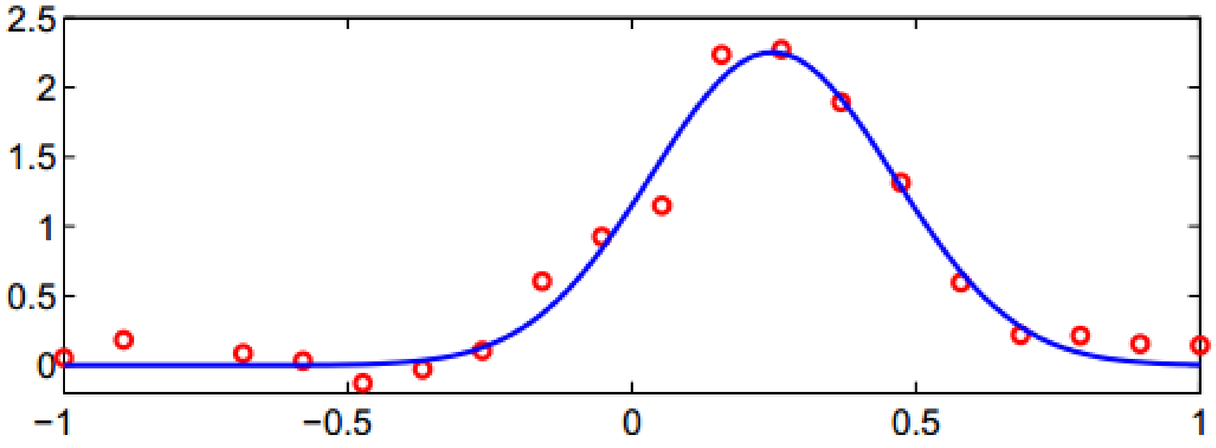
- Numerical stability
- Scalability
- Efficiency (look at Eigen for template magic)

Linear Solvers

- **Eigen**: http://eigen.tuxfamily.org/index.php?title=Main_Page
  - *Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.*
- Taucs: http://www.tau.ac.il/~stoledo/taucs/
  - *TAUCS is a C library of sparse linear solvers.*
- Umfpack
  - *UMFPACK is a set of routines for solving unsymmetric sparse linear systems of the form Ax=b, using the Unsymmetric MultiFrontal method (Matrix A is not required to be symmetric)*
- cuSPARSE: http://docs.nvidia.com/cuda/cusparse/index.html
  - *The cuSPARSE library contains a set of basic linear algebra subroutines used for handling sparse matrices, running on the GPU using Nvidia CUDA*
- Many more
  - https://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries

Non-Linear Least Squares

- Find solution that minimizes the sum of squared residuals
  - $f(x) = \sum r_i(x)^2$
  - r_i non linear with respect to x
- Ex: Fitting a Gaussian model



- $M(x,t) = x_1 e^{-(t-x_2)^2/(2x_3)^2}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
- Here: $r_i(x) = y_i - M(x, t_i)$

Non-Linear Least Squares

- $\underset{x}{\operatorname{argmin}} f(x) = ||F(x)||_2^2$

Gradient Descent (1st order):

- $x_{k+1} = x_k - t \cdot \nabla f(x_k)$

Newton's Method (2nd order):

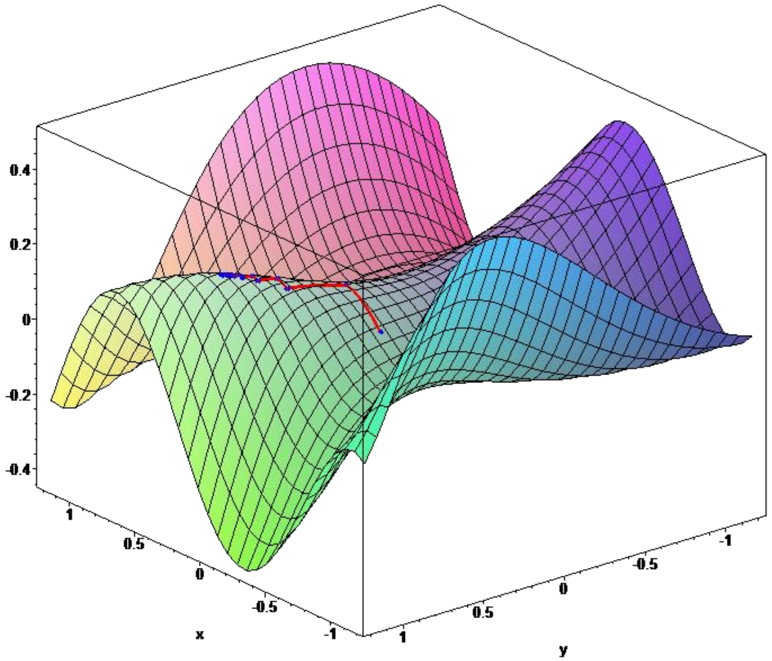
- $x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$

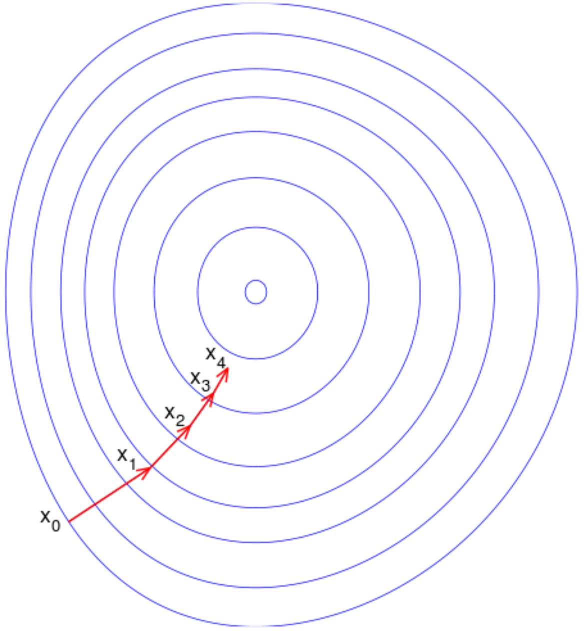
Non-Linear Least Squares: GD

Gradient Descent (1st order):

$$-\ x_{k+1} = x_k - t \cdot \nabla f(x_k)$$

- $\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$

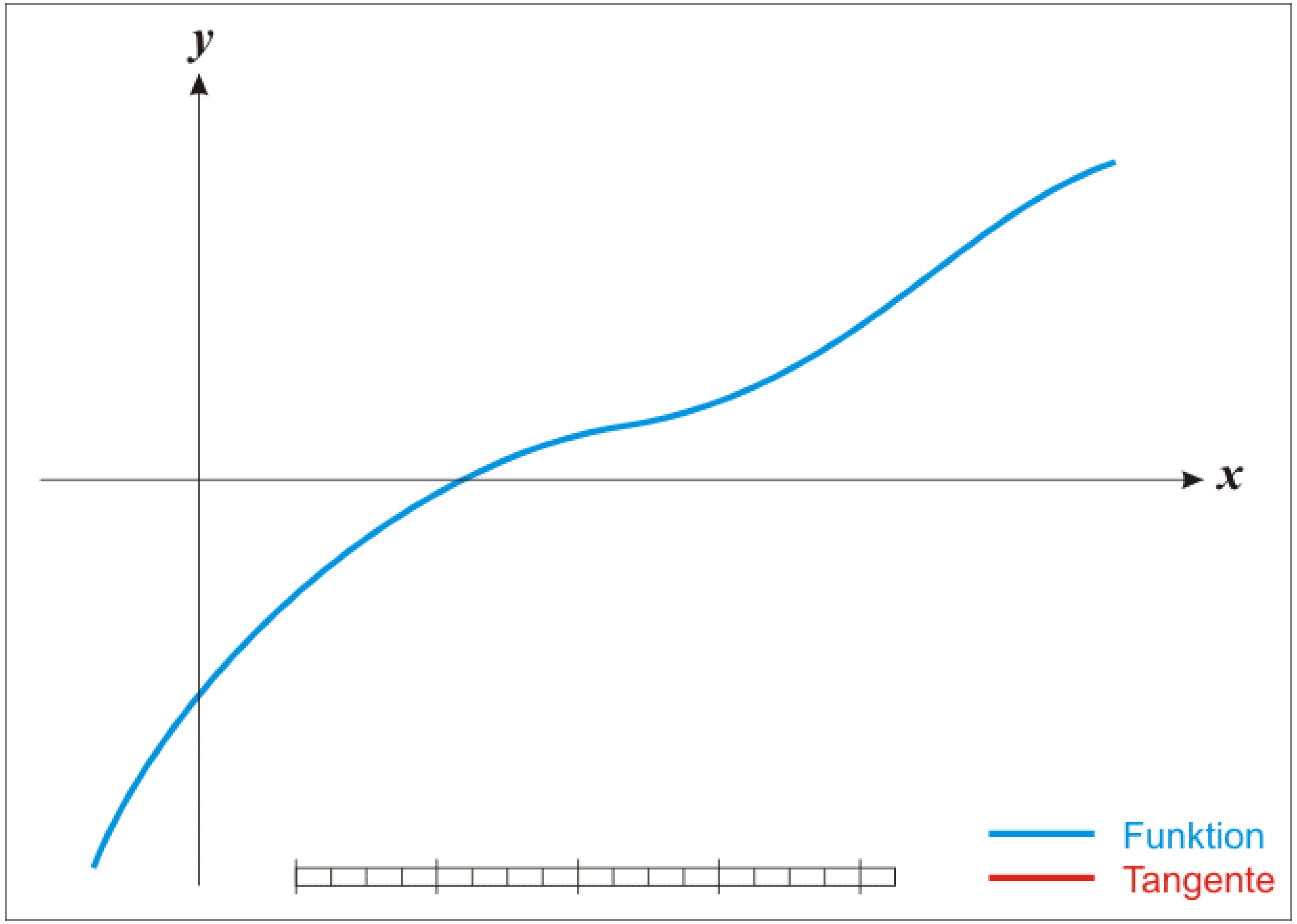




- Need to compute partials
- Need to determine step size
  - Line search
  - Momentum (i.e., track history)

Non-Linear Least Squares: Newton (root finding)

Root finding: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$



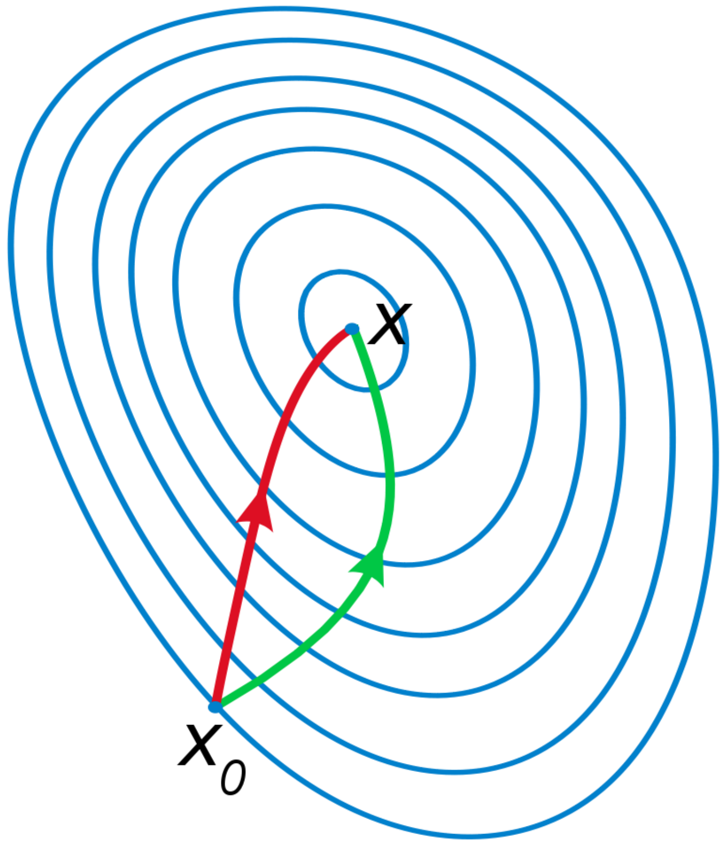
Non-Linear Least Squares: Newton

Newton's Method (2nd order):

– $x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$

- In 1D
  - Root finding: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
  - Optimization (find root of derivative)

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$



Newton (red) uses curvature information, and takes a more direct path than GD (green)

Non-Linear Least Squares

- Jacobian: $J_F(x) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}$ #residuals, #variables

  btw. $\nabla f(x) = 2 \cdot \left(J_F(x)\right)^T F(x)$

- Hessian: $H_f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$ #variables, #variables

  $H_f(x) = J_{\nabla f}(x)^T$

Non-Linear Least Squares: Gauss-Newton

- $x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$
  - 'true' 2nd derivatives are often hard to obtain (e.g., numerics)
  - $H_f \approx 2J_F^T J_F$
- Gauss-Newton (GN):

$$x_{k+1} = x_k - [2J_F(x_k)^T J_F(x_k)]^{-1} \nabla f(x_k)$$

- Solve linear system (again, inverting a matrix is unstable):

$$2\big(J_F(x_k)^T J_F(x_k)\big)\underbrace{(x_k - x_{k+1})}_{\text{Solve for delta vector}} = \nabla f(x_k)$$

Non-Linear Least Squares: Gauss-Newton

- $\underbrace{2\left(J_F(x_k)^T J_F(x_k)\right)}_{\boldsymbol{A}} \cdot \underbrace{(x_k - x_{k+1})}_{\boldsymbol{X}} = \underbrace{\nabla f(x_k)}_{\boldsymbol{b}}$

- Solve $Ax = b$
  - Could do matrix-free: applyJTJ, evalJTF

Non-Linear Least Squares: Gauss-Newton

- Solve $Ax = b$

- Common in our research:
  - Use Pre-conditioned Conjugate Gradient Descent (PCG)
  - Easy to parallelize; e.g., on GPUs

PCGInit Kernel

$\mathbf{r}_0 = -2\mathbf{J}^T\mathbf{F}$

$\mathbf{M}^{-1} = 1/\mathrm{Diag}(2\mathbf{J}^T\mathbf{J})$

`evalJTF()`

$\mathbf{p}_0 = \mathbf{M}^{-1}\mathbf{r}_0$

$\alpha_{n_0} = \mathrm{reduce}(\mathbf{r}_0^T\mathbf{p}_0)$

$\boldsymbol{\delta}_0 = 0$

`for i = 0 to num_linear_iterations:`

PCGStep1 Kernel

$\mathbf{g}_k = 2\mathbf{J}^T\mathbf{J}\mathbf{p}_k$

`applyJTJ()`

$\alpha_{d_k} = \mathrm{reduce}(\mathbf{p}_k^T(\mathbf{g}_k))$

PCGStep2 Kernel

$\alpha_k = \alpha_{n_k}/\alpha_{d_k}$

$\boldsymbol{\delta}_{k+1} = \boldsymbol{\delta}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k(\mathbf{g}_k)$

$\mathbf{z}_{k+1} = \mathbf{M}^{-1}\mathbf{r}_{k+1}$

$\beta_{n_k} = \mathrm{reduce}(\mathbf{z}_{k+1}^T\mathbf{r}_{k+1})$

PCGStep3 Kernel

$\beta_k = \beta_{n_k}/\alpha_{n_k}$

$\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$

$\alpha_{n_{k+1}} = \beta_k$

$\mathbf{F}$	Vector of original energy terms
$\mathbf{J}$	Jacobian matrix of $\mathbf{F}$
$\mathbf{r}_k$	Residual in the k-th iteration
$\mathbf{M}$	Pre-conditioner (remains constant)
$\mathbf{p}_k$	Decent step in the k-the iteration
$\alpha_k = \alpha_{n_k}/\alpha_{d_k}$	Step size in k-th iteration
$\boldsymbol{\delta}_k$	Vector of PCG unknowns in iteration k

Application-specific *solver routines*

Non-Linear Least Squares: Levenberg

- Levenberg
  - "damped" version of Gauss-Newton:

$$(2J_F(x_k)^T J_F(x_k) + \lambda \cdot I) \cdot (x_k - x_{k+1}) = \nabla f(x_k)$$

Tikhonov regularization

  - The damping factor λ is adjusted in each iteration ensuring:

$$f(x_k) > f(x_{k+1})$$

    - if not fulfilled increase λ

    →Trust region

→"Interpolation" between Gauss-Newton (small λ) and Gradient Descent (large λ)

Non-Linear Least Squares: Levenberg-Marquardt

- Levenberg-Marquardt (LM)
  - Extension of Levenberg:
    $$\left(2J_F(x_k)^T J_F(x_k) + \lambda \cdot diag\left(J_F(x_k)^T J_F(x_k)\right)\right) \cdot (x_k - x_{k+1}) = \nabla f(x_k)$$
  - Idea: Instead of a plain Gradient Descent for large λ, scale each component of the gradient according to the curvature.
    - Avoids slow convergence in components with a small gradient

Non-Linear Least Squares: BFGS / L-BFGS

- BFGS (Broyden-Fletcher-Goldfarb-Shanno)
  - Quasi-Newton method

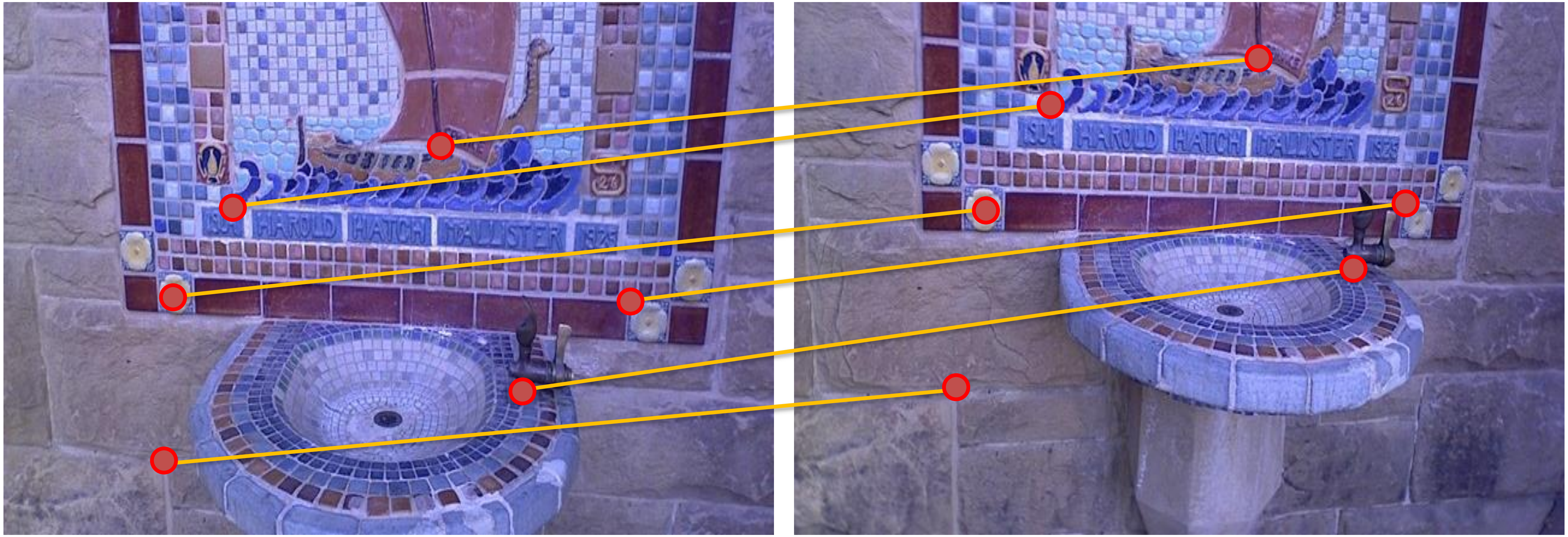
$$B_k \cdot (x_k - x_{k+1}) = \nabla f(x_k)$$

  - Approximation of the Hessian using rank-1 updates in each iteration:

$$B_{k+1} = B_k + \alpha \cdot uu^T + \beta \cdot vv^T$$

  - In practice, instead of approximating B_k, directly approximate B_k^{-1}
- L-BFGS (Limited-memory BFGS)
  - Approximation of BFGS

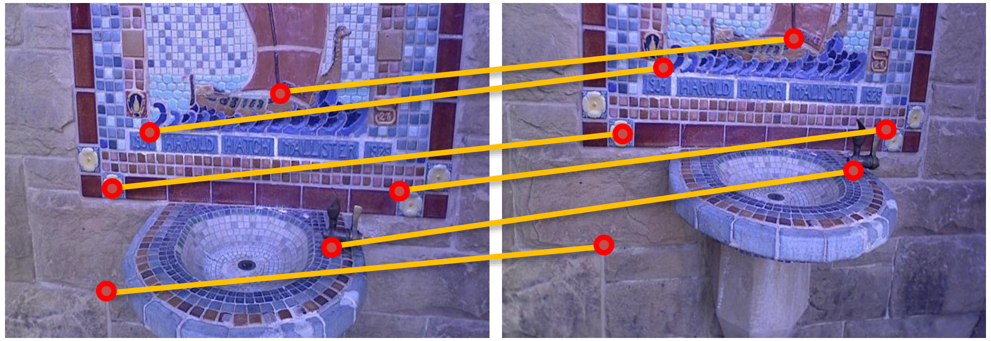
Back to 3D Reconstruction



$$E_{keypoint}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

Back to 3D Reconstruction

- Important: Don't merge residuals!
  - 2-Norm notation might be misleading



$$\underbrace{E(T)}_{\text{6-DoF Transforms}} = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \underbrace{\left\| T_i \overset{\text{correspondences}}{p_{ik} - T_j p_{jk}} \right\|_2^2}_{\substack{\text{These are 3 residuals each (for x, y, z)!} \\ \text{-> also 3 rows each in the Jaccobian}}}$$

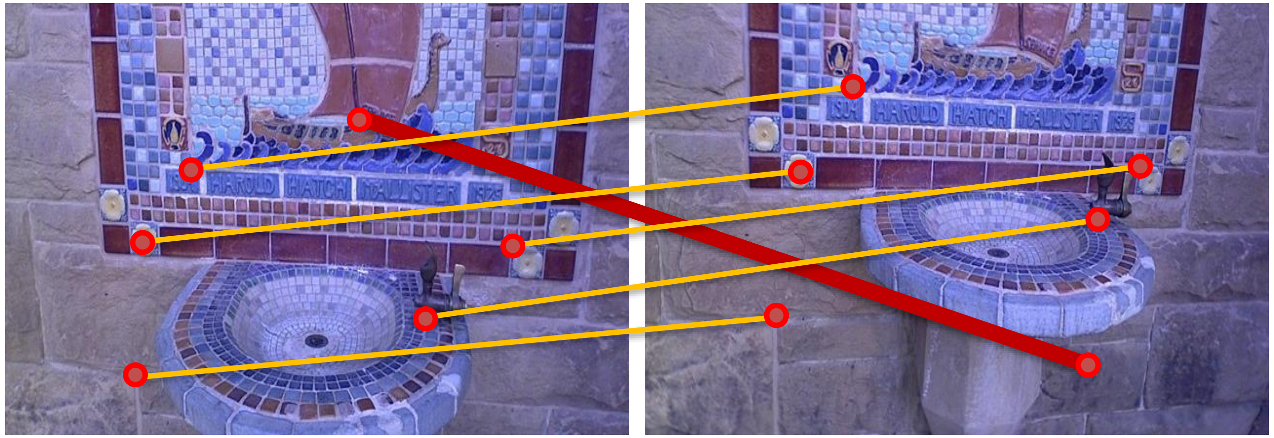
Handling Outliers

$$E_{keypoint}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \left\| T_i p_{ik} - T_j p_{jk} \right\|_2^2$$

Handling Outliers: Robust Optimization

- RANSAC (essentially trial and error)

- Lifting Schemes:
  - Good results
  - Costly to optimize



- Robust norms:
  - L-1
  - p-Norms
  - Huber Norm

Robust Optimization: Lifting Schemes

- $f(x) = \sum r_i(x)^2$
  - A single outliers kills the energy due to quadratic terms…
  - Introduce helper weights to weigh down outliers
  - Use regularization term to avoid trivial solution

- $f_{robust}(x, w) = \sum w_i^2 r_i(x)^2 + \lambda_{reg} \sum (1 - w^2)^2$

Many alternatives for 'lifting kernel'

  - Ideally, at the end of opt. all outliers are $w = 0$, inliers $\mathrm{w} = 1$

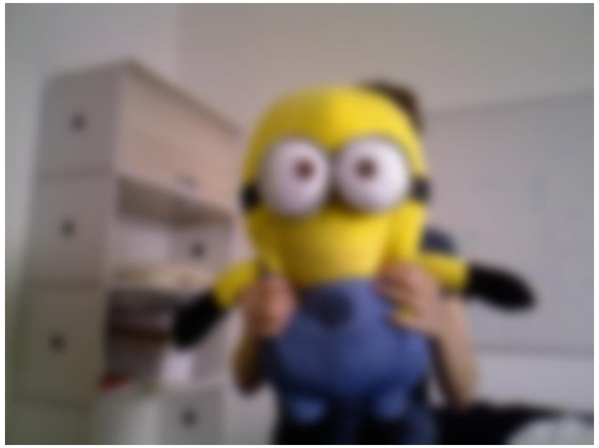
Iteratively Reweighted Least Squares (IRLS)

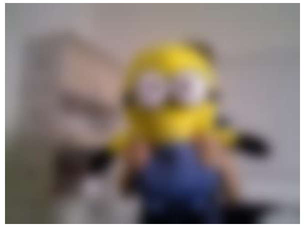
- $f(x) = \sum |r_i(x)|^p$
- $x^* = \underset{x}{\operatorname{argmin}} f(x) = \underset{x}{\operatorname{argmin}} ||F(x)||_p^p$
- Map to L2 problem for each iteration
- Iteratively solve $f(x) = \sum \underbrace{w_i}_{\text{Fixed for the current iteration}} r_i(x)^2$ and $w_i = |r_i(x)|^{p-2}$

Local vs Global Minima?

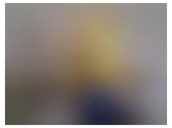
- Convexification

- Make energy landscape smoother and convex!
  – Smoothing
  – Coarse-to-fine strategies over unknowns
  – Coarse-to-fine strategies over residuals





RGB-alignment
is good example!



Performance / Efficiency Considerations

- Jacobian: $\mathrm{J_F}(\mathrm{x}) = \underbrace{\left.\begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}\right\}}_{\text{\#variables}} \text{\#residuals}$

Gauss-Newton:

$$2(J_F(x_k)^T J_F(x_k)(x_k - x_{k+1}) = \nabla f(x_k)$$

PCGStep1 Kernel

$\boxed{\mathbf{g}_k = 2\mathbf{J}^T\mathbf{J}\mathbf{p}_k \quad \texttt{applyJTJ()}}$

$\alpha_{d_k} = \mathrm{reduce}(\mathbf{p}_k^T(\mathbf{g}_k))$

- Sparsity of J
- How many unknowns?
- How many residuals?

How to apply JTJ?

(JTJ)p vs. JT(Jp)

  – Directly affects dimensions of J and JTJ

Computing Derivatives

- Numeric Derivatives
- Automatic Differentiation
- Symbolic Differentiation

Numeric Derivatives

- $\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$

- Forward Differences
$$\frac{df(x)}{dx} \approx \frac{f(x+h)-f(x)}{h}$$
- Central Differences
$$\frac{df(x)}{dx} \approx \frac{f(x+h)-f(x-h)}{2h}$$

- Easy to implement -> good for debugging
- Slow and numerically unstable

Automatic Differentiation: Dual Numbers

- $f(x) = x^2$
- Choose infinitesimal unit e, such that $e \neq 0$ but $e^2 = 0$
  - Dual number (similar to complex numbers)
- $f(10 + e) = (10 + e)^2 =$
  $100 + 2 \cdot 10 \cdot e + e^2 =$
  $100 + 20 \cdot e$

Try it out!

This is zero

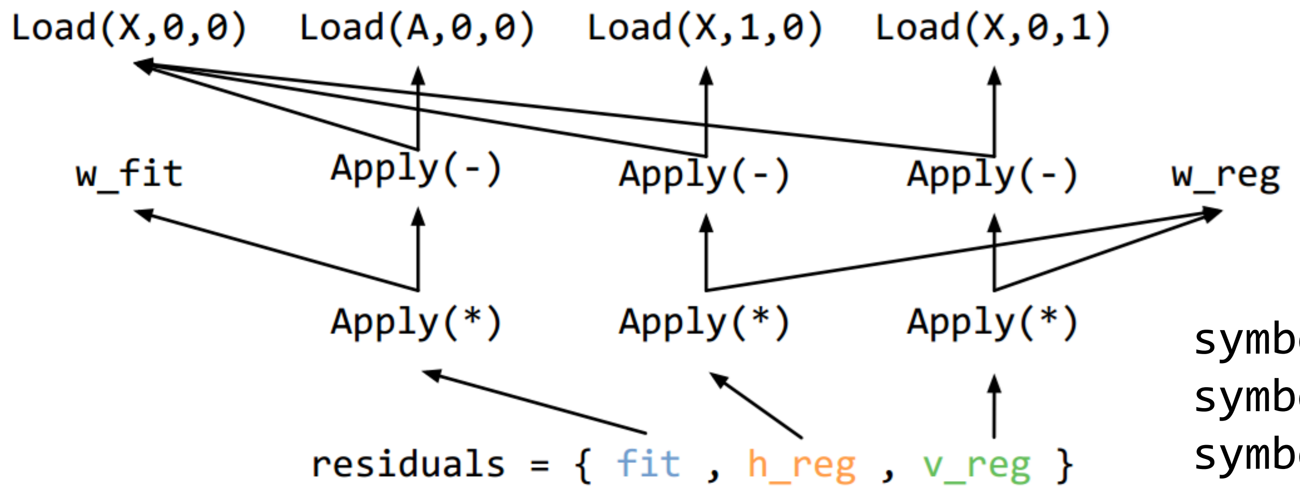
This is $\frac{df}{dx}$

Automatic Differentiation: Dual Numbers (‘Jets’)

```
template <typename T, int N>
struct Jet {
	…
	template<typename T, int N> inline
		Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
		Jet<T, N> h;
		h.a = f.a * g.a;
		h.v = f.a * g.v + f.v * g.a;
		return h;
	}
	T a;  // The scalar part.
	Eigen::Matrix<T, N, 1> v;  // The infinitesimal part.
};
```

Symbolic Differentiation

- For instance, D* [Guenter 07]
- Analyze compute graph at compile time!
  – Can simplify / fuse terms efficiently
  – Optimal solution is NP-Complete (but many heuristics)



Could of course also do on whiteboard ☺

```
symbolic_derivative(h_reg, X(1,0)) --> -w_reg
symbolic_derivative(h_reg, X(0,0)) -->  w_reg
symbolic_derivative(v_reg, X(0,1)) --> -w_reg
```

Non-linear Solvers

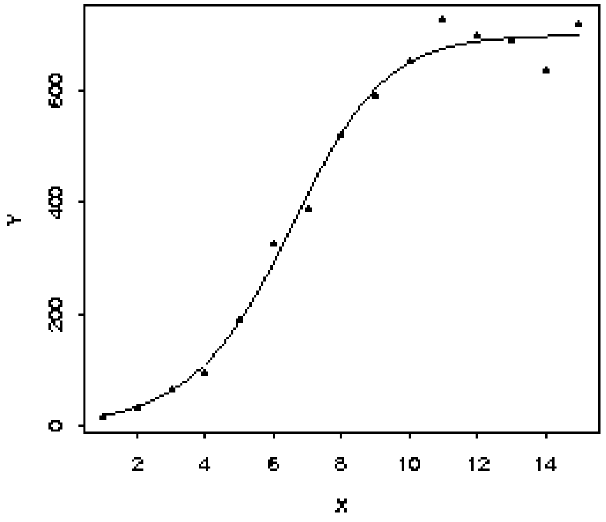
- **Ceres**
  - Uses Eigen as backend for linear solves (has also its own PCG)
  - Automatic differentiation using dual numbers ("jet.h")
- Alglib
  - Numerical differentiation or hand-provided
- Symbolic solvers
  - Maple
    - Good for derivations
    - Not so great simplification / code conversion

Introduction to Ceres

```
struct Rat43CostFunctor {
  Rat43CostFunctor(const double x, const double y) : x_(x), y_(y) {}

  template <typename T>
  bool operator()(const T* parameters, T* residuals) const {
    const T b1 = parameters[0];
    const T b2 = parameters[1];
    const T b3 = parameters[2];
    const T b4 = parameters[3];
    residuals[0] = b1 * pow(1.0 + exp(b2 -  b3 * x_), -1.0 / b4) - y_;
    return true;
  }

  private:
    const double x_;
    const double y_;
};


CostFunction* cost_function =
      new AutoDiffCostFunction<Rat43CostFunctor, 1, 4>(
        new Rat43CostFunctor(x, y));
```

$$y = f(b_1, b_2, b_3, b_4) = \frac{b_1}{\left(1 + e^{(b_2 - b_3 \cdot x)}\right)^{\frac{1}{b_4}}}$$

Note the templates!



Connection to Deep Learning

- Deep Learning uses stochastic Gradient Descent
- Backpropagation
- But no second order solvers

- True gradient is hard to compute for large training sets
  - Needs stochasticity
  - There is also theory why that helps with local minima
    - Theory: many local minima are equivalent in performance even though weights are different
- Stochasticity does not seem to well with 2^{nd} order solvers
  - There are attempts… don't seem to work so well

Connection to Deep Learning

- Operate on compute graphs
- Backpropagation of applying the chain rule
  – Keep track of derivatives
- Deep Learning frameworks typically support autodiff
  – E.g., Autograd in torch
  – I.e., implement only forward pass in layer, autodiff does the rest

Connections to Other Optimizations

- Hard- and inequality constraints
  - Lagrange multipliers
  - ADMM (Alternating Direction Method of Multipliers)
  - PD (Primal Dual)
- Gradient-free methods:
  - Monte-Carlo Methods
  - Metropolis Hastings
  - Genetic and evolutional solvers
- Differential equations

How do we solve these non-linear terms?

- Bundle Adjustment or RGB-D Bundling

$$E_{re-proj}(\boldsymbol{T}, \boldsymbol{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left\| x_{ij} - \pi_i(T_i \cdot X_j) \right\|_2^2$$

$$E_{keypoint}(T) = \sum_{i,j}^{\#frames} \sum_{k}^{\#corresp.} \| T_i p_{ik} - T_j p_{jk} \|_2^2$$

How do we solve these non-linear terms?

- Frame-to-frame alignment (RGB-D case)
- $E_{frame-to-frame}(T) = \sum_k \left\| p_{ik} - Tp_{jk} \right\|_2^2$
- How to align two RGB-D frames?
  – ICP!

Administrative

- Reading Homework:
  - Ceres Documentation: http://ceres-solver.org/automatic_derivatives.html
  - Research on RANSAC for correspondence finding
- Next week:
  - Rigid Surface Tracking & Reconstruction

Administrative

See you next week!