Lecture 1: Basics

Capture Devices

Passives:

+ work indoors + outdoors
− Need Compute
− Features needed

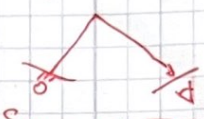
→ RGB

→ Stereo and Multi-View

Active:

+ No features needed
− Fails outdoors
− Sensitive to scattering.

→ ToF (time of flight)
→ Structured light
→ LiDAR.
− slow
+ very precise



↳ Same as ToF but needs precise calibration between projector and sensor

Stereo Matching: Sparse vs. Dense Matching

Wider dist. between cameras: ↑ distortion, ↑ occlusions
Smaller dist between cameras: ↑ error

Feature extraction vs. Not.

Search window: smaller = more noise
larger = less details

Correspondence Failures: 1) Non-Lambertian surfaces ⇒ specular reflection (is not diffuse)
2) Occlusions/repetitions
3) Textureless Surface

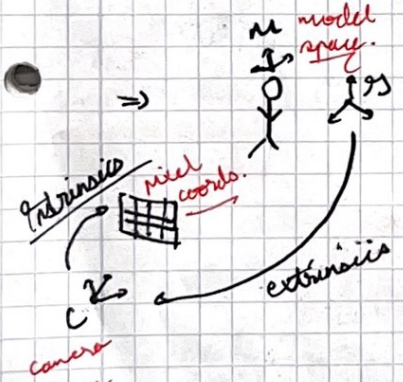
Camera Concepts

→ Intrinsics: → Focal Length
→ Principal Point.
→ (Skew)
→ (Distortion)

$$K = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

γ: shear coefficient
c_x, c_y: principal point or optical center.
f_x, f_y: focal length.

→ Extrinsics: R, t.



$\Rightarrow \quad \Rightarrow K \vec{x_c} = z_c \vec{u}$

Models other than Pinhole

→ Radial + Tangential Distortion: use approximate values using camera calibration.

Fish eye lens.

↳ image plane not parallel to lens

Lecture 2: Surface Representations

Polygonal Meshes:

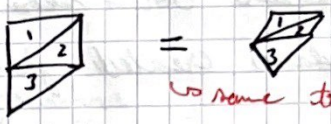
obviously efficient rendering.

→ Error in shape representation: $O(h^2)$ ↝ i.e. doubling the # of (2x) edges gives $(\frac{1}{2})^2$ reduction in error.

→ Allows for recursive subdivisions

↳ i.e. higher resolution

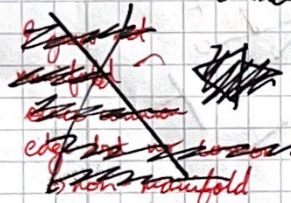
Topology (triangulation) vs. Geometry

Topologically equivalent ⇏ Geometrically Different	 ↝ same triangulation
Geometrically equivalent ⇏ Topologically Different	 ↳ different triangulation but same rectangle.

Manifolds

*Triangle mesh is called manifold if:

(i) Intersection of two triangles are (+ have):



and/or { * empty ↝ i.e. no area.
* common vertex
* common edge.

here intersection spans the area of small triangle ⇒ not empty. ⇒ Non-manifold

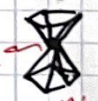
edge ab has three adjacent triangle ⇒ Non-manifold

(ii) Edges have:
* One adjacent triangle → border edge
* Two adjacent triangles → inner edge

(iii) Vertices:
* Build a single open fan → border vertex
* Build a single closed fan → inner vertex.

vertex connected to two open fans ⇒ non-manifold

vertex is connected to TWO closed fans ⇒ Non-manifold



Mesh Data Structures

(i) Shared Vertex

Shared Vertex data structure

* OFF Obj STL etc. ↝ PLY allows for binary storage
* list v → list f.
  - - - {vertices
  - - - 3 faces.

v ...
v ...
f ...

* (ii) Half-Edge Data Structure ↝ only accepts manifold representations (since stores adjacent edges)

↳ store vertices + faces + edges (i.e. opposite edge, adjacent edge etc.)

Useful for geometric computations ↝

Surfaces

(i) Explicit surfaces: $f: \mathbb{R}^2 \to \mathbb{R}$ i.e. give me x, y and I give height ↳ for terrain in games

(ii) Parametric surfaces:

Constructed from Bezier Curves → Bezier patches

Used in CAD. & design + construction etc.

Bezier curve → bezier patch: $f(u,v): \mathbb{R}^2 \to \mathbb{R}^3$

(iii) Constructive Solid Geometry:

Boundaries created from boolean operations:

(iv) Implicit Surfaces: → SDF
→ Heat
→ Density

Defining Implicit Surfaces:

(i) Hoppe method: $f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \vec{n}_p$ ~ normal of point p

for any x ↳ for nearest p. (from point cloud e.g.)

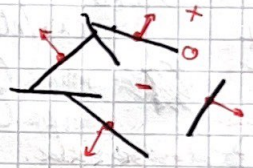
→ local method

→ bad with outliers

→ Creates Voronoi diagram which defines closest points

→ Piecewise linear

→ Discontinuous along edges of voronoi



(ii) RBF: Radial Basis Functions:

→ global method

→ Better at handling outliers

We assume smoothness ⇒ $f(\vec{x}) = \sum_i^n \alpha_i \underbrace{\varphi_i(\vec{x})}_{\text{some } C^2 \text{ function}} + bx + d$

To solve: $f(\vec{p_i}) = 0$

$f(\vec{p_i} + \varepsilon \vec{n_p}) = \varepsilon$

$f(\vec{p_i} - \varepsilon \vec{n_p}) = -\varepsilon$

} our constraints

Unknowns: $\begin{matrix} a_1 \\ \vdots \\ a_n \end{matrix} \Big| \begin{matrix} b_1 \\ b_2 \\ b_3 \end{matrix} \Big| d$ $\quad n + 3 + 1 = n+4$

Linear ⇒ SVD to solve for α

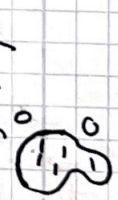
(iii) Poisson Reconstruction

$\nabla \cdot \nabla f(\vec{x}) = \nabla \cdot \vec{N}$ ~ normal vector field

is basically divergence of all our normals

divergence of our $\nabla f(\vec{x})$ ↳ gradient field

for poisson we don't use SDF



Rendering Implicit Surfaces

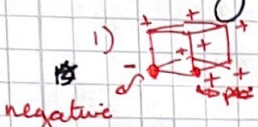
(i) Ray Casting

1) Ray Marching : * Apply fixed step length.
* Linearly Interpolate if zero crossing.

2) Sphere Tracing : * Dynamic Steps
* Stop if distance below threshold.

(a Go-to method usually.

(ii) Marching Cubes : 1) Determine zero crossings for all grids cells.



2) If zero-crossing use Lookup table to triangulate

3) Linearly interpolate triangulation

2) Use which vertex negative as ID in Lookup Table.

3) ~~[illegible]~~ Find Triangulation

4) Linearly interpolate triangulation

Lecture 3: 3D reconstruction Methods

Feature Detection

(i) FAST Detectors: for pixel $p(x,y)$ if ≥ 12 pixels are brighter (in circular kernel for example) ⇒ feature point

(ii) Harris Corner Detectors: if we have I_x, I_y: (partial derivatives wrt. x, y: $I(u,v) - I(u+1,v)$ etc.)

$$H = \begin{pmatrix} \sum_{c \in w} I_x^2, & \sum_{c \in w} I_x I_y \\ \sum_{c \in w} I_x I_y, & \sum_{c \in w} I_y^2 \end{pmatrix}$$ ~ Hessian of some image patch W. basically

we can do SSD for e.g. Harris but not robust.

we find λ_1, λ_2 of H: $\lambda_1 \sim \lambda_2$ + small ⇒ flat region
$\lambda_1 >> \lambda_2$ OR $\lambda_2 >> \lambda_1$ ⇒ Edge region
$\lambda_1 \sim \lambda_2$ + large ⇒ corner region.

Trick to find λ_1, λ_2 w/o SVD: $R = \det(M) - \alpha (\text{trace}(M))^2$
$= \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$ ⇒ properties of det + trace

if $\lambda_1 \sim \lambda_2 \sim 0$ ⇒ $R \sim 0$ i.e. flat region
if $\lambda_1 >> \lambda_2$ ⇒ $\lambda_1 \lambda_2 \sim 0$ ⇒ $R = -\alpha \lambda_1^2$ ⇒ edge
if $\lambda_1 \sim \lambda_2$ ⇒ $\lambda_1^2 \sim \alpha \lambda_2^2$ ⇒ corner.
(α is usually 0.04 - 0.06)

if: $|R|$ small → flat region
$R < 0$ → edge region
$R > 0$ → corner region
} cool trick actually :)

(iii) SIFT Detectors: finds features that are scale invariant (and rotation invariant) ~ i.e. applies multiple scalings + blur and finds features not affected + finds dominant rotation direction to compare features. (+ filter (usually HARRIS))

Correspondence Matching

(i) Simple SSD per feature: shitty + not robust

(ii) SIFT Descriptor: after SIFT filters optimal features: (i) takes 16-sub blocks neighbourhood around feature point, (ii) rotates neighbourhood in preferential direction, (iii) Calculates 8-bin orientation histogram per sub-block ⇒ 8·16 = 128 dimensional feature vector ~ per feature I guess.

(iii) ConvNet: Basically siamese (shared weights) network inputting some image/volume patch from two views and outputs 1/0 depending on match or not.

(iv) SURF, ORB etc.

Main 3D recon methods:

(i) Visual Hull Carving: If we have: → n cameras
→ we know each camera's intrinsics
→ we know each camera's extrinsics
→ we segment subject in each camera

Carve our voxel grid based on each camera segmentation ⇐

(ii) Structure from Motion (SfM): Given: → Set of images
→ Features + correspondences for each image

Bundle Adjustment

Re-projection energy. optimizes over T_i^* + X_j^*

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

- x_{ij}: 2D coords of feature point j in image i
- $\pi_i(T_i X_j)$: 3D X_j transformed to camera frame i: $T_i X_j$ then projected to camera i.
- i: images
- j: feature points

Bundle Adjustment ⇐ { Goal: Find 3D locations X_j + Find Camera poses T_i.

* Unknowns: → $6 \cdot (m-1)$ ↝ Extrinsics of all cameras except first (since identity)
→ $3 \cdot n$ ↝ Feature points in 3D space.
→ k (optional:) ↝ Camera Intrinsics per camera.
* Constraints: → $2 \cdot m \cdot n$ ↝ all feature points in all images in 2D space.

$\Rightarrow \quad 2 \cdot m \cdot n \geq 6(m-1) + 3n + k.$

(iii) Multiple View Stereo (MVS): Given: → Set of images
→ Poses of images T_i
→ 3D locations of features X_j

Goal: → Dense Point Cloud.

* We can do frame pair matching + vote
* Global optimization is NP-Hard
* Plug point clouds to surface fitting ↝ e.g. poisson surface fitting

↳ Usually Pipeline: 2D images → SfM → MVS → Reconstruction → 3D mesh.

(iv) SLAM: ↝ smaller compute than SfM + MVS.

* Basically real-time SfM + MVS
* Different than Bundle Adjustment since no simultaneous offline global optimization of all poses + 3D locations.
* Types: RGB { 1) ORB SLAM, 2) LSD SLAM. } can also either use feature points for alignment OR directly input
RGB-D { 3) KinectFusion

Frame to Frame OR Frame to Model

SLAM usually has Loop-Closure Problems

RGB-D Bundling

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

- p_{ik}: corr. k in frames i
- p_{jk}: backprojected corr. k in j in global X_j
- square but for j
- Value each projected corr. in i: 1) backproject in i, 2) transform to 3D w/ T_i, 3) to j camera frame, 4) Project$_2$ to 2D, 5) Depth Value in j, 6) backproject in j, 7) transform to j w/ T_j, 8) to original i camera frame T_i

$$E_{depth}(T) = \sum_{i,j} \sum_{k} \| (p_{ik} - T_i^{-1} T_j \pi_j^{-1}(D_j(\pi_j(T_j^{-1} T_i p_{ik})))) \cdot n_{ik} \|_2$$

$$E_{color}(T) = \sum_{i,j} \sum_{k} \| \nabla I(\pi_i(p_{ik})) - \nabla I(\pi_j(T_j^{-1} T_i p_{ik})) \|_2^2$$

- $\nabla I(\pi_i(p_{ik}))$: color gradient of original pixel in frame i
- $\nabla I(\pi_j(T_j^{-1} T_i p_{ik}))$: color gradient of projected pixel in frame j.

Lecture 4: Optimisation Methods:

Linear Least Squares

* Means the model is linear in parameters

⇒ $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \dots$

↳ here X_2 can be x^2, $\log x$, $\frac{1}{x}$ doesn't matter still linear since linearly multiplying by β_2.

* Simplified to: $Ax = b$ ↳ overdetermined usually

* Usually solved as normal equation: $A^T A x = A^T b$

Solvers: ~~s~~ of linear least squares

→ Iterative Solvers: conjugate
1) Preconditioned Gradient Descent. ~~s~~ most common iterative solver.
2) Gauss-Seidel Iteration
3) Jacobi Iteration

→ Direct Solvers:
1) QR, LU - Decomposition
2) SVD
3) Cholesky

related to each other I think

Non-Linear Least Squares

* No longer linear in params: e.g. $Y = \beta_0 X^{\beta_1}$ ~~s~~ β_1 not linearly multiplied

Solvers: we're trying to solve $\underset{x}{\operatorname{argmin}} f(x) = \|F(x)\|_2^2$

→ First Order Solvers: Gradient Descent ofc.

⇒ $x_{k+1} = x_k - t \cdot \nabla f(x_k)$ ~~s~~ momentum / line search standard BS.

→ Second Order Solvers:

(i) Newton's Method: $x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$ ~~s~~ comes from taylor btw: $f(x+t) \approx f(x) + \nabla f \cdot t + \frac{1}{2} H_f \cdot t^2$

(ii) Gauss-Newton: Approximation of Newton:

Assume $x_k \in \mathbb{R}^n$

$H_f \approx 2 J_F^T J_F$ ($n \times n$, $n \times n$) ⇒ $x_{k+1} = x_k - [2 J_F^T J_F]^{-1} \nabla f(x_k)$

⇒ $f'(x+t) = \nabla f + H_f \cdot t = 0$

⇒ $t = H_f^{-1} \nabla f$.

so: $x_{k+1} = x_k - t$.

~~s~~ residuals where $F \in \mathbb{R}^n$

linear system btw. ??

$\Rightarrow 2 J_F^T J_F \underbrace{[x_k - x_{k+1}]}_{x} = \underbrace{\nabla f(x_k)}_{b}$ (A)

⇒ solve with iterative ~~linear solvers~~ like preconditioned conjugate gradient descent (PCG)

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

↳ Tikhonov regularization i.e. damping factor for each term in J s.t. $\lambda \uparrow$ if ~~$f(x_k) > f(x_{k-1})$~~ $f(x_k) < f(x_{k-1})$

↳ $\lambda \downarrow \to$ GN

$\lambda \uparrow \to$ Gradient Descent.

if applied for linear system converges in one step

BRUNNEN

(iv) Levenberg-Marquardt (LM): faster convergence this way than Levenberg

$$x_{k+1} = x_k - \left[2J_F^T J_F + \lambda \cdot \mathrm{diag}(J_F^T J_F)\right]^{-1} \nabla f(x_k)$$

more sophisticated Levenberg since each λ scaled by curviture of (diagonal is independent parts of curve)

(v) BFGS / LBFGS:

$$x_{k+1} = x_k - B_k \nabla f(x_k)$$ Basically approximation of Hessian.

where $B_{k+1} = B_k + \alpha \cdot u u^T + \beta \cdot v v^T$

In Practice we use Limited Memory BFGS (LBFGS)

Outlier Handling:

? How do we convexify our problems?

course to fine over residuals

(i) RANSAC: basically trial & error

(ii) Lifting Schemes:

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

our lifting term that penalizes $w \to 0$ (inliers are $w = 1$)

(iii) Robust Norms: usually hard to optimize

$\Rightarrow$ Iteratively Reweighted Least Squares

$f_{robust}(x) = \sum w_i r_i(x)^2$ and $w_i = |r_i(x)|^{p-2}$ $\Rightarrow$ for some p-norm.

assuming we wanted $\arg\min_x f(x) = \arg\min_x \|F(x)\|_p^p$

Derivative Types:

(i) Numeric derivatives: $\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h}$ $h \sim$ s.t. take very small h.

→ Easy to implement ⇒ good for debugging

→ Slow and numerically unstable

(ii) Automatic Differentiation: Use dual numbers: $e \neq 0$, $e^2 = 0$

Example: $f(x) = x^2$ $\left.\frac{df(x)}{\partial x}\right|_{x=10}$? $\Rightarrow f(10+e) = 100 + 20e + e^2$ ($e^2 = 0$)

$\Rightarrow \left.\frac{df}{dx}\right|_{x=0} = \frac{20}{e}$

(iii) Symbolic Differentiation:

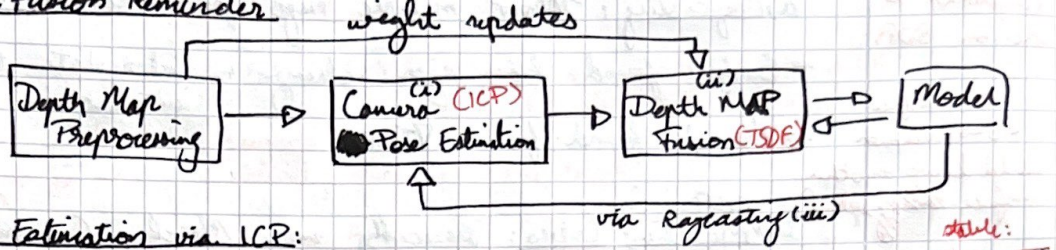
Basically no numerical solutions (Maple :))

Other Solvers: 1) Inequality Constraints: Lagrange multipliers etc.

2) Gradient-Free: Monte Carlo etc.

Lecture 5: Rigid Surface Tracking + Recon (i.e. KinectFusion revision)

KinectFusion Reminder



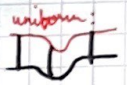
(i) Pose Estimation via ICP:

Basic Steps of ICP: iterate
1) Selecting source points
2) Find Correspondences
3) Weigh Correspondences
4) Reject Outliers
5) Assign Errors metric
6) Minimize error for pose estimate

Either:
- → Use all points
- → Uniform subsampling
- → Random sampling
- → Stable sampling

stable:

uniform:



life course to find registration for concretification of problem.

Finding Correspondences in ICP:

a) Nearest neighbour ↝ FLANN etc.
b) Projective correspondences ↝ did in da project i.e. (only works if we have depth values)

* We can improve matching via only checking compatible points:

$p_s \rightarrow \pi(K T_d T_s (\pi^{-1}(D(p_s), K)))$ → my correspondence in other frame.

a) Color Compatibility
b) Normal Compatibility
c) Pruning ↝ e.g. at border.
d) Reject angle and distance outliers

Errors in ICP:

a) Point to Point: $\sum_i \| (T p_i - q_i) \|_2^2 = E(T)$ ↝ had closed form linear solution!?

b) Point to Plane: $\sum_i \| (T p_i - q_i) \cdot n_i \|_2^2 = E(T)$ ↝ usually solved w/ LM if not linearized

idea here is energy lowest if $p_i \& q_i$ on the same surface w/ normal n_i

↳ can be linearized in $\theta \approx 0$ assuming small rotations ($\cos\theta \approx 1$) → Cholesky / SVD

* Symmetry ICP: $(p - q) \cdot (n_p + n_q)$

(ii) SDF / TSDF: (Implicit Representations):

- Basically representation of distance to zero-crossing
- TSDF: just applies truncation band i.e. if > 1 from zero-crossing / < -1 ⇒ $= 1 / -1$ everywhere

Volumetric Fusion: ↝ Surfel representation also exists: point + normal + radius

current TSDF based on our current depth value

Running average concepts: $D_{i+1}(x) = \frac{W_i(x) D_i(x) + w_{i+1}(x) d_{i+1}(x)}{W_i(x) + w_{i+1}(x)}$

$W_{i+1} = W_i(x) + w_{i+1}(x)$

weights can either be 1 everywhere OR closer to camera ↑ weight (uncertainty weighting)

tsdf value at voxel i currently (i.e. distance to the surface)

weights

(iii) Surface Rendering

(important to note on SDFs:
=0 → surface
<0 → unknown (could have another surface behind)
>0 → free space.)

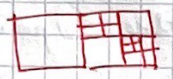
a) Raycasting: Throw multiple rays from camera.
- Surface found from sign change + interpolate to find intersection
→ Normal found from TSDF

b) Marching Cubes (posterior): Basically use Marching Cubes table based on zero-crossing.

Follow Ups to KinectFusion:

1) Extended KinectFusion: + easy to implement
+ unlimited spatial volume.
- cannot reintegrate previous regions.

2) Hierarchical Fusion: + fast lookups
+ efficient storage (high local res)
- costly updates (insertions/removals)
(= basically uses sparse voxel octrees →)



3) Voxel Hashing: + extremely fast updates (insertions/removals) O(1)
+ efficient storage
- lookup slower ⇒ raycasting not as fast
+ cheap streaming

Loop Closure Follow Ups:

1) Elastic Fusion: → Surfel based
→ Detects loop closure by localization
→ Non-rigidly warp to close loop.

2) Bundle Fusion: → Added sparse energy term for pose estimation
→ Includes surface de-integration if pose had wrong estimate

Lecture 6: Deformation & Non-rigid Surface Tracking

Mesh Deformations

- $E = \lambda_{fit} E_{fit}(V) + \lambda_{reg} E_{reg}(V, X)$ ← standard; we pick non-rigid regularisers which affects behaviours of our mesh.

$E_{fit}(V) = \sum_{i \in C} \|c_i - v_i\|_2^2$ ← minimize distance between vertex and target; target pos.

1-ring fan:

Regularisers Types

← Our goal for regularisers: → Smooth Global changes → Global changes preserve local detail → Should be zero in undeformed state.

i) Laplacian Surface Editing:

$E_{reg}(V, X) = \sum_i \sum_{j \in N} \|(v_i' - v_i) - (v_j' - v_j)\|_2^2$

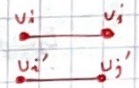
↳ our fan neighborhood (1-ring) ↳ i.e. deformation of v_j' should be very similar to 1-ring neighbour v_i'

ii) As Rigid as Possible (ARAP):

- $E_{reg}(R, V) = \sum_i \sum_{j \in N} \|(v_i - v_j) - R_i(v_i' - v_j')\|_2^2$ ← rotation with euler α, β, γ; means that after some R edge

done per fan over all vertices

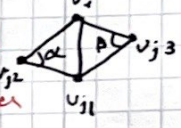
$v_i' v_j' \overset{!}{=} v_i v_j$



after some Rotation.

* we could add weight to give bigger priority to closer neighbours: $w_{ij} = \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$

as $\alpha_{ij} \uparrow$ $\cot\alpha_{ij} \downarrow$ ⇒ if v_{ij2} closer than v_{ij1} means lower weight for v_{ij1}



- Solving: Unknowns: $3n + 3n$ ← 3 positions per vertex; 3 euler angles per vertex

Constraints: $3n$ ← per position; valence + $3c$ ← average # fan neighbours; ↳ from E_{fit}

→ We can do Levenberg Marquardt (non-linear)
OR
→ Flip-flop optimization: 1) Assume $v_i' v_j'$ constant and solve for R
2) Assume R constant and solve for $v_j' v_i'$
⇒ linear problem can be solved with SVD / Procrustes.

iii) Embedded Deformation (ED): relaxed version of ARAP

$E_{reg}(M, V) = \sum_i \sum_{j \in N} \|(v_i - v_j) - M_i(v_i' - v_j')\|_2^2$

$E_{rot} = \sum_i Rot(M_i)$, $Rot(M) = (c_1 \cdot c_2)^2 + (c_1 \cdot c_3)^2 \dots + (c_{c_1} \cdot c_{c_1 - 1})^2 + \dots$

- Solving: Unknowns: $12n$

⇒ quartic problem to optimize

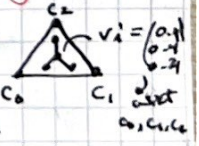
↳ i.e. we want M to resemble R but allow for more DoF in our deformation (shear etc.)

Main goal: dimensionality reductions

Deformation Proxies

← instead of deforming mesh ⇒ computationally expensive; deform proxy

i) Cages: linear combination of vertices wrt. cage.
↳ we use barycentric coordinates for that.



⇒ we deform cages and by proxy our vertices using barycentric coordinates.

we can apply ARAP/ED/Laplacian here.

ii) 3D Grids : ~~[illegible]~~

iii) Skeletons : → hierarchical : Body → left shoulder → left hand → head.

e.g. blend skinning

Called linear blend skinning or

- Each vertex where is a linear combination of bone matrices : $v_i = \sum_u w_{iu} M_u v_i'$; w_{iu} weight for each bone ; M_u each bone has a matrix

iv) Deformation Graphs : assign a region/neighborhood a control node. is similar to cage but instead we have deformation zone.

Non-Rigid Tracking is find correspondences then ICP.

needs predefined mesh.

+ Non-Rigid ICP: → more degrees of freedom. e.g. ARAP / ED.

$E = \lambda_{point} E_{point} + \lambda_{plane} E_{plane} + \lambda_{colors} E_{colors} + \lambda_{reg} E_{reg}$.

$\lambda_{point} E_{point}$: from regular point to plane ICP

$\lambda_{colors} E_{colors}$: i.e. lower energy if colors match

E_{point} : $\sum_i \| v_i - T v_i' \|_2^2$

E_{plane} : $\sum_i [(v_i - T v_i') \cdot n_{v_i}]^2$ → forces v_i' to stay on the plane

$n_i \perp (v_i' - v_i) \Rightarrow E = 0$

E_{colors} : $\sum_i [I(\pi(v_i)) - I'(\pi(T v_i))]^2$ of v_i

→ lower energy if colors match.

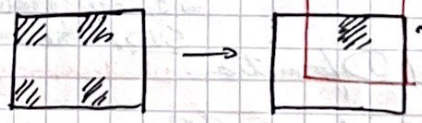
E_{reg} : ARAP.

* Important hyperparameter-thresholds for non-rigid ICP:
  1) distance
  2) angle
  3) Occlussion
  4) Viewing angle

Improves with more cameras.

* Correspondence Association : → for speed.

e.g. 1) Projective correspondences every k-the element
2) Kernel around k-th sample for minimum reduction

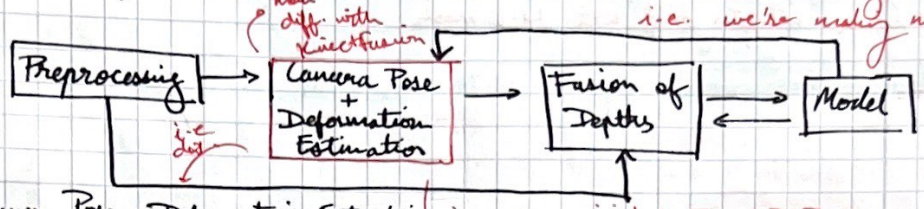


→ find best match within kernel.

Lecture 7: Non-Rigid Tracking and Reconstructions

General Concepts:

→ here we're tracking AND fusion (non-rigid) i.e. we're making mesh.



* For Camera Pose + Deformation Estimation: i.e. non-rigid ICP + ARAP etc.

- Iterate:
  1) Deform SDF
  2) Find Isosurface → usually with marching cubes since raycasting hard for non-rigids
  3) Find Correspondences
  4) Optimize deformations → non-rigid ICP with some non-rigid regularizers
  5) Integrate RGB-D data to model

* Main Methods discussed: → we are marching cubes here.

(i) **Dynamic Fusion**:
- → Uses Deformation Graphs (ED)
- → Non-rigid ICP
- → Volumetric Fusion.

(ii) **Volume Deform**:
- → Uses regular deformation grids + ARAP
- → Non-rigid ICP using sparse SIFT matches.
- → Volumetric Fusion

More stable + Better Texture than this.

(texture) We can increase resolution based on grid. This means higher stability compared to dynamic fusion.

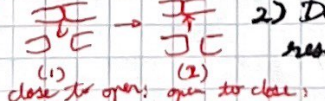
(iii) **Real-time Geometry & Motion Reconstruction**:
- → Deformation Graphs (ED)
- → Non-rigid ICP (dense depth + dense color) → instead of SIFT.
- → illumination correction

(iv) **Fusion 4D**: → way more complex.
- → 8 Depth Maps (16 IR cameras)
- → Key volumes: multiple poses → i.e. allows for topological changes (say removing coat etc.

(v) **Motion 2 Fusion**:
- → Basically faster Fusion 4D: 200 FPS
- → ED Graphs
- → Key Volumes
- → Multiple Depth Cameras.

* **Motion 2 Fusion had 2 novelties**: 1) 2 way backward and forward non-rigid alignment → i.e. solving topology problem.

solves it by making reference to data then data to reference:



2) Detail Layer: they increased resolution of grid at surface (viewing ray)

(vi) **LookinGood**: → increase / improve reconstructions using deep architectures

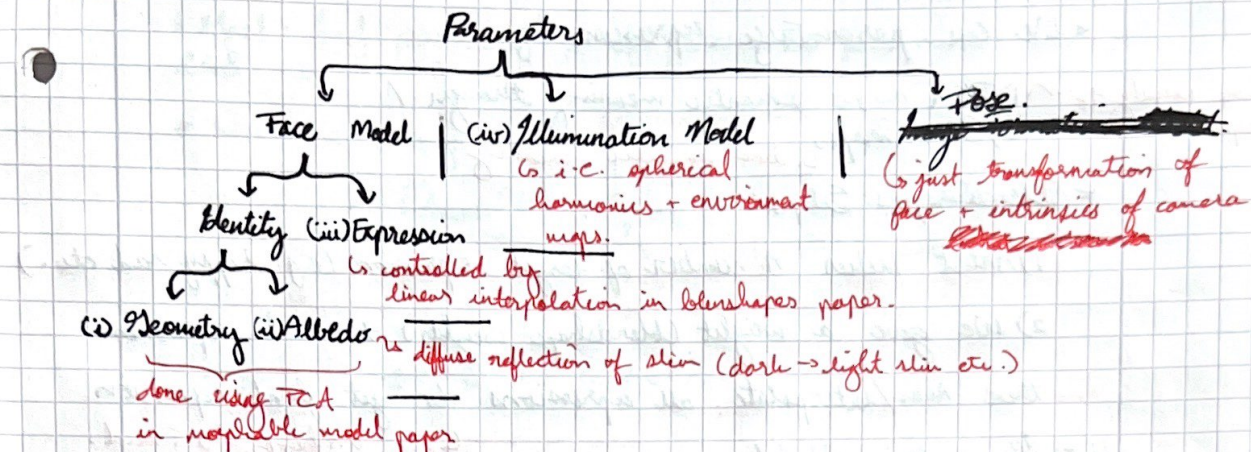
(vii) **DeepDeform**: → Find better correspondences than SIFT for best tracking
→ Use Siamese network that shares weights for two RGBs to give you correspondences.

The two RGBs are the source (query points) and target where correspondences need to be found.

⇒ For non-rigid reconstructions, as accumulates drift if tracking is wrong.

Tracking with a pre-defined mesh mitigates drift wrt. base mesh.

Lecture 8: Parametric Face Models:



(i) Identity: Geometry

* Here by geometry we mean structure of face (can include expressions)
* Steps: (According to face warehouse + morphable model)
  1) Scan a bunch of faces
  2) Fit topologically consistent template non-rigidly to faces
  3) Since all registration from some model we have 1 to 1 correspondences
  ⇒ 3) Create one massive vector of all positions of vertices per face
  4) Find PCA-basis for all the vectors (i.e. all faces)

(usually PCA means no semantic meaning)

⇒ $M_{geo}(\alpha) = a_{id} + E_{id}\,\alpha$ (linear parameter. this also means we get average face like this!)
(average shape) (PCA basis that we found.)

PCA notes:

eigenvectors of the

* The principle components are the n largest eigenvalues of covariance matrix

(also means direction of largest variance in our data or line with minimal dist. to data.)

Covariance:
$$cov(X,X) = \frac{1}{N-1}\sum_i \bar{X}_i \bar{X}_i^T$$
where $\bar{X}_i = X_i - \bar{X}$ is average $\frac{X_i}{c}$

(ii) Identity: Albedo

* Here we mean light/dark skin
* Steps exactly the same as Geometry but done independently of Geometry!

average albedo — PCA basis we found

$M_{alb}(\beta) = a_{alb} + E_{alb}\,\beta$ (linear parameter.)

(iii) Expression:

* We can parametrize expressions by:

done usually to compress data.

1) PCA ↝ no semantic meaning though :/
2) BlendShapes ↳ less semantic meaning

* BlendShapes: Steps:

1) Artist makes n number of target expressions (e.g. happy, sad etc.)

2) We give a weight (blendshape weights) to each expression then sum/interpolate all expressions to get final expression
↳ e.g. 0.5 happy + 0.5 sad.

(iv) Illumination Model:

Steps

↳ Environment Maps
↳ Spherical Harmonics ↝ parametrization of environment maps

*) Environment maps place the object a cube/sphere map of some panorama scene

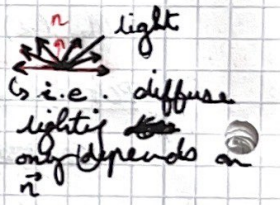
2) Project lighting onto object (which is in center of sphere/cube)

↳ Assumptions: → Distant light
→ No scattering

* Spherical harmonics parametrize / reduce dimensionality of environment maps. (orthogonal basis)

⇒ we can linearly interpolate basis spheres for different lighting

↳ assumptions: → Lambertian Surface:
→ Distant Smooth lighting



↳ i.e. diffuse lighting only depends on $\vec{n}$

Lecture 9: Face Tracking and Reconstruction

* Goal: find parameters p^* of parametric face model to fit RGB 2D image.

* We make energy term: → for energy minimization

$$E(P) = E_{dense}(P) + E_{sparse}(P) + E_{reg}(P)$$

Energy of color and geometry of mask wrt. P

Energy Correspondences terms wrt. P

regularization of P

$$E_{dense}(P) = E_{geom}(P) + E_{col}(P)$$

point to point ↔ point to plane ↔ color.

$$E_{geom}(P) = \sum_i \| M_i(P) - D(\pi(M_i(P))) \|_2^2 + \left[(M_i(P) - D(\pi(M_i(P)))) \cdot n_i \right]^2$$

point to point

point to plane

summed over all vertices

mask vertex

correspondent depth value at projected mask vertex in camera frame

dot product of mask mesh normal.

$$E_{col}(P) = \sum_i \| M_i(P)_c - I(\pi(M_i(P))) \|_2^2$$

colors of mask at vertex i (in mesh)

color of pixel of at projected mask vertex in camera frame.

$$E_{reg}(P) = \sum_k \left(\frac{\alpha_k}{\sigma_{id,k}} \right)^2 + \sum_l \left(\frac{\beta_l}{\sigma_{alb,l}} \right)^2 + \sum_m \left(\frac{\delta_m}{\sigma_{exp,m}} \right)^2$$

α, β, δ are parameter vectors of id, alb, exp & centered at zero ofc. The summands done for each dimension

done for all correspondences

$$E_{sparse}(P) = \sum_j \| \pi(M_j(P)_m) - C_j \|_2^2$$

2D coordinates of projected mask in camera frame

2D coordinates of correspondences in 2D image

$\Rightarrow P^* = \operatorname{argmin}_P E(P)$, solved with non-linear least squares.

↳ Gauss Newton or LM solver.

$\Rightarrow P_{n+1} = P_n - (J_F^T J_F)^{-1} J_F^T \cdot F$ ↝ Usually done Rough → Dense pyramid

Partial Derivatives of 2D image wrt. P:

↳ needed for E_{color} + E_{reg}

↳ probably the same but not sure :/

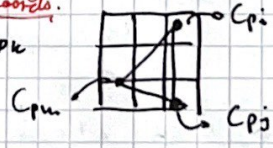
1) For each color pixel:

$$C_{pixel} = \alpha C_{pi} + \beta C_{pj} + \gamma C_{pk}$$

colors at pi etc.

Barycentric coords.

linear interpolation of vertex color.



2) record triangle vertex index, barycentric coords α, β, γ.

3) $\Rightarrow \frac{\partial C_{pixel}(P)}{\partial P_j} = \alpha \frac{\partial C_{pi}(P)}{\partial P_j} + \beta \frac{\partial C_{pj}}{\partial P_j} + \gamma \frac{\partial C_{pk}}{\partial P_j}(P)$

* This parameter update done:

Synthesis → Parameter Update → Synthesis

* Can be done without depth via multiple views
  ↳ energy jointly optimized for 6 images

* Energy optimization dependent on:
  → Input Data (RGB vs. RGB-D)
  → Face Model + Illumination (course vs. dense)
  → Run-time requirements, hardware
  → Application

* Two main papers using dis:
  → Real-time expression transfer for facial reenactment
    ↳ for two live videos
  → Face2Face
    ↳ expression to RGB video

Lecture 10: Body & Hand Tracking

→ Parameterization for consistent topology of hand + body

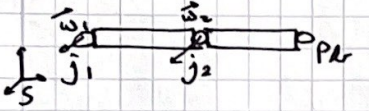
→ Parameters of body: $M(\vec{\theta}, \vec{\beta}, \vec{u} \mid \phi)$ → hyperparameters.
pose of body, shape, texture

CAESAR Dataset
→ 125 M 125 F
→ 74 markers.
→ Various ethnicities, age, weight.

→ Same idea as before: 1) average mesh + Non-rigid ICP
2) PCA of all mesh vectors.

SCAPE: → joint shape + pose learning.
→ blend shapes used first for poses

Skeletons Sidenote:



: each joint has a transformation matrix related to rotation by rodrigues formula.

$$\Rightarrow \boxed{P_S = \mathcal{G}_S(\vec{\omega}_1, j_1)\, \mathcal{G}_S(\vec{\omega}_2, j_2)\, P_B}$$

→ poses are given by $\theta = (\vec{\omega}_1, \dots \vec{\omega}_k)$ where $\mathcal{J} = (\vec{j}_1, \dots \vec{j}_k)$

Linear Blend Skinning: For any vertex $\vec{t}_i$ on mesh:

vertices are linear combinations of all joints ←

$$\boxed{t_i' = \sum_{k=1} w_{k,i}\, \mathcal{G}_k'(\theta, \mathcal{J})\, \bar{t}_i}$$

$w_{k,i}$: weight given to each joint; $\mathcal{G}_k'$: transformation of each joint

Corrective Blend Shapes:

Basically displace each vertex by $P = \begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 \\ \vdots & & \\ \Delta x_n & \Delta y_n & \Delta z_n \end{bmatrix}$ to simulate muscle compression :)

Non-rigid registration: $E = w_d E_d + w_s E_s + w_m E_m$

$E_d = \sum_i w_i\, \text{dist}^2(T_i v_i, D)$ — Data error.

$E_s = \sum \| T_i - T_j \|_F^2$ — Transformation Smoothness

$E_m = \sum \| T_{k,i}\, v_{k,i} - m_i \|^2$ — marker Errors.

Hand Models:

Taylor: Uses subdivision model → recursive coarser mesh to get:
→ C_2 continuity
⇒ Fast LM convergence

Issues with Model Generation + Fitting:

- → Expensive Scanning setup.
- → Neutral Pose similar but not identical
- → Ambiguities in fitting w change shape or expression?
- → Drift in non-rigid registration