Concepts

3D Representations:

Voxels: binary (or probabilistic) occupancy grid. Cubic memory

Point clouds: 3D points (unstructured) with attributes

Polygon Mesh: set of vertices and faces (color = interpolation of face vertices' color)

Parametric surface: set of control points (splines)

Implicit surface: signed distance field. Implicit: $x+y=0$ Explicit: $f(x)=-x$

Camera spaces

World space → (Camera space) [R|t] → Model space

Camera space → (Pixel coordinate) K → Normalized device coordinate (NDC)

Normalize [-1, 1]

Extrinsics

$$E = \begin{bmatrix} R & t \\ 0\,0\,0 & 1 \end{bmatrix} \qquad R \in \mathbb{R}^{3\times3} \in SO(3) \qquad t \in \mathbb{R}^3$$

$SO(3)$: $|R| = 1$, $RR^T = I$

Camera Pose = $\begin{pmatrix} R_x & U_x & L_x & X_c \\ R_y & U_y & L_y & Y_c \\ R_z & U_z & L_z & Z_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (right, up, look, position)

$E = Pose^{-1}$. This definition is relative. Could also be $E = Pose$

6 DOF: 3 for R (in $so(3)$ lie algebra representation) + 3 for t

$R = R_z(\gamma) R_y(\beta) R_x(\alpha)$

Intrinsics

$$K = \begin{pmatrix} f_x & r & m_x \\ 0 & f_y & m_y \\ 0 & 0 & 1 \end{pmatrix}$$

f: focal length, dist from pinhole to image plane, aspect ratio: f_x / f_y. In pixel units

m: principal point, dist to proj. center $(w/2, h/2)$. In pixel units

r: axis skew

$f_x = f\frac{w}{W}$ $\quad f_y = f\frac{h}{H}$

f = focal length in mm

w, h = pixels

W, H = sensor width/length in mm

Calibration

Radial Distortion: fish-eye effect.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Tangential Distortion: barrel effect

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Stereo Matching

Estimate depth with 2 rgb camera.

- Rectification: make epipolar lines parallel to horizontal axis of image planes.
  Epipolar constraints, essential matrix.
- Search: find match (with 2D feature) along epipolar lines

$$SSD = \sum_{(u,v)} [I_{left}(i,j) - I_{right}(i,j)]^2.$$

Possible features: SIFT, SURF...

Window size: small → noisy
large → few detail
search time also correlated to window size

Baseline (distance between cams): small → less accurate (small disparity results in large error)
large → hard to find matches (occlusion)

Devices

Passive:

Stereo (Multi) camera: rely on 2D feature and expensive matching / finding

Active:

ToF: IR spectrum (fails outdoor), sensitive to scattering, indirect light

Structured light: IR spectrum (fails outdoor), need calibration between projector and sensor

Laser scanner / Lidar: slow and precise or fast and sparse.
precision due to trivial feature matching.

Surface Representations

Polygonal Mesh

Approximation error $O(h^2)$

Arbitrary topology (triangulation)

Efficient rendering (GPU rasterization)

It's a manifold if:

- Any intersection of 2 triangles → Empty / Common edge / Common vertex
- Edges have either 1 or 2 adjacent triangles
- For a vertex the adjacent triangles build a single open or closed fan

topology (triangulation), geometry (shape)

Formats: obj, off, ply...

Half-edge data structure: easy geometric query (directed edges)

Explicit Surface

$f(x,y): \mathbb{R}^2 \to \mathbb{R}$ $\quad S(x,y) = (x, y, f(x,y))$

only one hight value per (x,y) pair
e.g. height map

Parametric Surface

$f(u,v): \mathbb{R}^2 \to \mathbb{R}^3$ freeform surface with set of control points
eg: Bezier, splines

Constructive Geometry

surface created by boolean operations on primitive solids. OP: $\cup, \cap, -, \ldots$

Implicit Surface

$f(x,y,z): \mathbb{R}^3 \to \mathbb{R}$ SDF (signed distance function)

Hesse normal form: $f(x,y,z) = \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \vec{p}\right)\vec{n} = 0$

Given sample points find SDF?

Hoppe

$f(\vec{x}) = (\vec{x} - \vec{p}) \cdot \vec{n}_p$ $\vec{p}$ the nearest neighbor of $\vec{x}$ in the given sample points

↳ $f(\vec{x})$: Distance to the tangent plane of $\vec{p}$

piecewise linear (linear boundaries, Voronoi diagram), not smooth.
Can't handle noise, outliers...

RBF

theo: complex functions can be approximated as the sum of simple scaled and translated kernel functions $\varphi(x)$

$f(\vec{x}) = \sum_i d_i \cdot \varphi_i(\vec{x}) + \vec{b} \cdot \vec{x} + d$ $\varphi_i(\vec{x}) = \|\vec{p}_i - \vec{x}\|^3$ biharmonic basis function

i: sample points

params (unknowns): d_i (n), $\vec{b}$ (3), d (1) } n+4

constrains: $f(\vec{p_i}) = 0$ (n), $f(\vec{q_i}) = \pm\epsilon$ (n) } 2n

$\vec{q_i} = \vec{p_i} \pm \epsilon \cdot \vec{n}_{p_i}$

$A \cdot \vec{x} = \vec{b}$ overdetermined

$(A^T A \cdot \vec{x} = A^T \vec{b})$

$$\begin{bmatrix} \cdots & p_i & \cdots \\ \cdots & q_i & \cdots \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ b \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ \cdots \\ \pm\epsilon \end{bmatrix}$$

Can handle noise and outliers, but only works with small point set (slow).
A workaround is to partition point cloud into parts and fit individually.
Or coarse-to-fine...

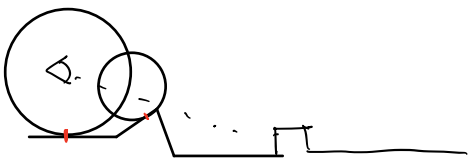
Rendering SDF

Ray Marching

Fixed steps, linear interpolation when crossing occurs.

March forward to find intersection

Sphere Tracing



A type of ray marching for SDFs.
At each step solve for
$f(x,y,z)=0$ (max sphere radius).

Then we can take step of at least this value (safely). → Dynamic steps

Stops if distance is below some threshold.

Marching Cubes

Create a polygonal mesh from a SDF. In 3D each SDF cell has 8 vertices ⇒ check zero crossing in lookup table of $2^8 = 256$ combinations.

Vertex position can be adjusted with linear interpolation of SDF values.

Poisson Surface Reconstruction

Goal: obtain mesh (extracted from isosurface) from input sample points and normals. $O(n^2)$ in time and memory. Need dense point cloud.

Estimate indicator function χ

$\nabla\chi = \vec{V} \Rightarrow \Delta\chi = \nabla\cdot\nabla\chi = \nabla\cdot\vec{V}$ (Laplacian = divergence of gradient)

Vector field $\vec{V}$ is the surface integral (we don't know it before hand), but it can be approximated by the summation over patches (neighborhood).

$$\vec{V}(q) \approx \sum_{s\in S} |\mathcal{P}_s| \cdot F_{sp}(\vec{q}) \cdot s.\vec{N}. \qquad \vec{V}(q) = \nabla(\chi * F)(\vec{q}). \qquad F = \text{Gaussian filter}$$

Solve for $\sum_{o\in\mathcal{O}} \| \langle \Delta\chi, F_o \rangle - \langle \nabla\cdot\vec{V}, F_o \rangle \|^2 \Rightarrow \min_{x\in\mathbb{R}^{\text{dim}}} \|Lx - v\|^2$ Gauss-Seidel

Mesh reconstruction from χ_M

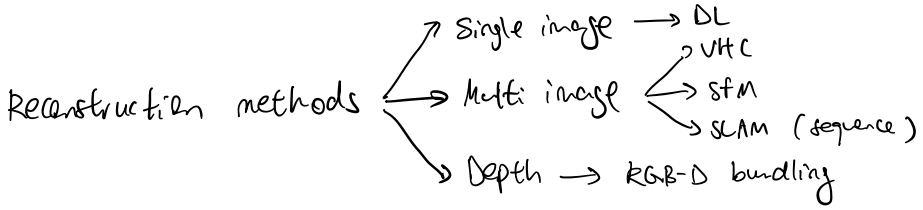
Evaluate χ of sample points and take average to find appropriate isovalue.
Perform marching cubes to extract surface (mesh).

Screened Poisson

$O(|P| \log |P|) \approx O(n^{1.5})$ complexity

$(\Delta - \alpha \tilde{I})\chi = \nabla \cdot \vec{V}$ α = screening weight

Reconstruction Methods



Single Image

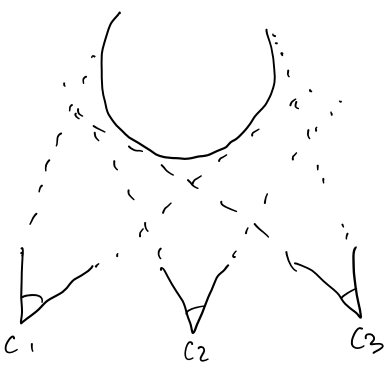
Extremely ill-posed problem, often need strong priors, guess shape from shading.

Hallucinate shape → deep learning

Multiple Image

Visual Hull Carving

Calibrated multi camera setup (known E,K)



Need object segmentation to identify foreground (silouette). Then with known (E,K) back-project object to 3D as a silouette cone. Intersect the cones from other views.

Apply smoothing kernel to improve quality.

Could also do with depth data by fusing views

Sensible to noise

Structure from motion (Sfm)

Construct mesh from images taken from different viewpoints.

1. Estimate camera params (E,K) and 3D points.

1.1 Find keypoints of each image

- FAST (features from accelerated segment test): (•p) comparare the intensity of p + th to the 16 nearby pixels forming the circle.

If more than n (usually 12) contiguous pixels are all greater or all smaller than Ip+th then p is a corner (keypoint).

- HARRIS corner detector: find corners in a 3×3 sliding window manner. $E(u,v)$ measures the sum of SSD in all 8 directions.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)] \approx [u,v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}.$$

→ eigenvectors give direction of greatest change in SSD

→ eigenvals (λ_1, λ_2) can be used to determine:

λ_1, λ_2 small ⇒ flat region
λ_1, λ_2 large ⇒ corner
λ_1, λ_2 differs ⇒ edge

1.2. Correspondence matching (feature descriptor)

Naively SSD. But ideally should be: scale, view, light invariant

- SIFT (scale invariant feature transform)

Detector:

Scale invariance ⇒ different octaves each one's size is half of prev.
DoG ⇒ progressive blur images (change σ) in a octave and compute difference of gaussian (DoG)
Laplacian of gaussian ⇒ a pixel is compared to it's neighbor (8), 9 from prev and 9 from next scale. If this pixel is maxima, then it's a potential keypoint.

Descriptor:

Scale, rotation, illumination invariance.
16×16 window around the keypoint which is then subdivided in 4×4 sub-blocks, each one with 8-bin orientation histogram ⇒ 4×4×8 = 128 dim
Matching ⇒ dot prod.

rot invariance: from all orientation extract keypoint's orientation
illu invariance: cap (clamp) the value in feature vector and normalize.

- ConvNet (pretrained weights, ResNet)

1.3 Bundle Adjustment

formulate the reprojection errors and minize it by solving the non-linear least square problem.

m images (cams), n total keypoints:

$6(m-1)$	extrinsics	}	params
$3n$	points		
possibly	intrinsics		

make sure $2mn > 6(m-1) + 3n + (\text{intrinsics})$

solve to obtain sparse point cloud and camera pos.

$2m \cdot n$	(2D)	} constraints

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

2. Multiview stereo (MVS)

With result from 1. (K, E, points), estimate dense point cloud.
Do dense stereo matching between frame pairs (rate for results)

3. Reconstruction

With dense point cloud from 2. estimate normals with PCA and do Poisson reconstruction to obtain mesh. Texture resulting surface

PCA for normals: with the neighbors of a point compute centered cov. matrix and perform eigen decomposition. Eigen vector corresponding to 3rd eigenvalue (ordered) is the surface normal

Simultaneous Localization and Mapping (SLAM)

Similar idea to sfm+MVS but online.
model is built incrementally by frame to frame tracking.
Bundle adjustment on keyframes only and use ORB features to save computing budget.

RGB-D Bundling

Important: here T's are camera-to-world matrix. (Extrinsics^{-1})

Depth data available, frame-to-frame tracking or frame to model (accumulate depth data into model, SDF). May have loop closure issue.

Sparse: $E_{bundle}(T) = \sum_{i,j} \sum_{k} \| \overbrace{T_i P_{ik}}^{\text{world}} - \overbrace{T_j P_{jk}}^{\text{space}} \|^2$ (i,j) frames, k correspondence, P_{ik} 3D point (depth from sensor)

Dense: $E_{depth}(T) = \sum_{i,j} \sum_{k} \| (p_k - T_i^{-1} T_j \pi_d^{-1}(D_j(\pi_d(T_j^{-1} T_i P_k)))) n_k \|^2$ k pixels, D_j depth map

$$E_{color}(T) = \sum_{i,j} \sum_{k} \| \nabla I(\pi_c P_k) - \nabla I(\pi_c (T_j^{-1} T_i P_k)) \|^2$$

Optimization

Least squares

Solves for $f(x) = \sum r_i(x)^2$ $\quad x^* = \underset{x}{\operatorname{argmin}}\, f(x) = \underset{x}{\operatorname{argmin}} \|F(x)\|_2^2$ $\quad f(x) = [r_1(x), \dots,]^T$

Linear Least squares

residuals are linear wrt parameters. Could be expressed as $Ax = b$ (input, residuals)

Normal Equation

if $Ax=b$ is overdetermined (# constraints > # params). solve by
$A^TAx = A^Tb \Rightarrow x = (A^TA)^{-1}A^Tb$

Iterative methods

(parallelizable but slow convergence)

- Jacobi iteration : $A = L+D+U$, $x_{k+1} = D^{-1}(b - (L+U)x_k)$
- Gauss-Seidel iteration : $A = L+U$, $x_{k+1} = L^{-1}(b - Ux_k)$ (sequential but fast convergence)
- Conjugate gradient descent : $x_{k+1} = x_k + \alpha_k p_k$ $\quad \frac{1}{2}x^TAx - b^Tx$ (PCG) $\quad \alpha_k, p_k$ dependant to residual r_k

$(L+U)x = b$;
$Lx + Ux = b$;
$Lx = b - Ux$;
$x^{k+1} = L^{-1}(b - Ux^k)$

Direct methods

- QR : $A = QR$, $Q^T = Q^{-1}$
- LU : $PA = LU$
- cholesky : $A = LL^T$
- SVD : $A = U\Sigma V^T$ $\quad (U\Sigma V^T)x = b$; $x = V\Sigma^{-1}U^Tb$

$PAx = Pb$; $LUx = Pb \Rightarrow Ly = Pb$; ($Ux = y$)
$y = L^{-1}Pb \Rightarrow x = U^{-1}L^{-1}Pb$

Non-Linear Least Squares

residuals non linear wrt params.

1st order

Gradient Descent : $x_{k+1} = x_k - t\,\nabla f(x_k)$

2nd order

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

Gauss-Newton: $Hf(x_k) \approx 2J_F^T J_F$ $\quad x_{k+1} = x_k - (2J_F^T J_F)^{-1} \nabla f(x_k)$ $\quad \underbrace{(2J_F^T J_F)}_{A}\underbrace{(x_k - x_{k+1})}_{x} = \underbrace{\nabla f(x_k)}_{b}$

Levenberg: $Hf(x_k) \approx 2J_F^T J_F + \lambda \cdot I$ $\quad \dots \quad J_F = J_F(x_k)$

Levenberg-Marquardt: $Hf(x_k) \approx 2J_F^T J_F + \lambda \cdot \mathrm{diag}(J_F^T J_F)$

BFGS: $Hf(x_k) \approx B_k$ $\quad B_k(x_k - x_{k+1}) = \nabla f(x_k)$ $\quad B_{k+1} = B_k + \alpha uu^T + \beta vv^T$. In practice directly estimate B_k^{-1}

L-BFGS $\Rightarrow$ approximation with limited memory

Handling Outliers

RANSAC: trial and error. For a pair of frames try different combination of correspondence (by only taking a subset) and see which gives the lowest error (outlier $\Rightarrow$ high error)

Lifting Scheme: optimize additional parameter w, which are weights for each residual term. Let optimization decide which are outlier by assigning low value (ideally 0 for outlier, 1 for inliers). Regularizer is added to avoid trivial solution.

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

Robust norms (IRLS: Iterative Reweighted Least Squares): e.g. L_1 is more robust to outliers than L_2, however hard to optimize. IRLS trick:

for any norm p: $F(x) = \sum |r_i(x)|^p$ map to L_2 in each iteration

$= \sum \underbrace{w_i r_i(x)^2}_{L_2}$, $\underbrace{w_i = |r_i(x)|^{p-2}}_{L_{p-2}}$ $\quad (p = p - 2 + 2)$

↳ changes at each iteration as x gets updated.

Efficiency and Convergence

To make the optimization smoother and more convex, first apply smoothing and adopt coarse-to-fine strategy.

Exploit sparsity of J and parallelize optimization with GPU for maximal efficiency

Derivatives

Jacobian: $J_F(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots \\ \vdots & \ddots \\ & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$ residuals (m), vars (n)

Gradient: $\nabla f(x) = 2 (J_F(x)^T) F(x)$ $\quad (\mathbb{R}^{m \times n})^T \mathbb{R}^{m \times 1} = \mathbb{R}^{n \times m} \mathbb{R}^{m \times 1} = \mathbb{R}^n$.

when only 1 residual $\Rightarrow J_F(x) = \nabla f(x)$. Thus, gradient is a special case of jacobian

Hessian: $H_F(x) = J_{\nabla F}(x)^T = \begin{bmatrix} \frac{\partial f}{\partial x^2} & \cdots \\ \vdots & \ddots \\ & \frac{\partial f}{\partial x_n^2} \end{bmatrix}$ vars(n), vars (n)

Ways to compute derivative:

Numerical Derivative

$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ easy to implement and debug but slow and unstable

Auto Diff

Ceres Dual Numbers.

Choose infinitesimal unit e, such that $e \neq 0$, $e^2 = 0$.
Let $f(x) = x^2$:
$f(10+e) = (10+e)^2 = 100 + 20e + e^2 = 100 + 20e \Rightarrow f'(10) = 20$.
verification: $f'(x) = 2x$, $f'(10) = 2 \cdot 10 = 20$

Symbolic Diff

Analyse compute graph at compile time, can simplify terms efficiently.
However optimal solution is NP-Complete

Connection to DL

In DL mainly stochastic gradient descent is used to train the network.
Done by backpropagation on compute graphs, often handled by frameworks like Pytorch. stochasticity is need for large data set and also helps with local minima.

Other Optimization methods

Inequality → Lagrange multiplier, Primal Dual

Gradient-free → Monte Carlo, Hill Climbing, Simulated Annealing, Genetic algorithm

Differential eqns

Rigid Tracking and Reconstruction

Goal: RGB-D tracking, move camera over time and incrementally build the scene. Would need to align point clouds

Aligning Point Clouds

Procrustes algorithm (correspondence)

Given point clouds X, Y, want: $\min_{R,t} \| Y - RX + t \|_F$

scaling: compute center for both point clouds. Scale objects to match the avg. distance from vertices to center of gravity in the target point cloud.

rotation: 1. Center $X, Y \Rightarrow X_c = X - \bar{x}$ $Y_c = Y - \bar{y}$ $\bar{x}, \bar{y}$ means

2. $\min \| Y_c - RX_c \|_F^2 \Rightarrow \min tr(Y_c^T Y_c - Y_c^T X_c R^T - (X_c R^T)^T Y_c + (X_c R^T)^T (X_c R^T))$

($\|A\|_F^2 = tr(A^T A)$; $Y_c^T Y_c$ doesn't affect R; $R^T R = I$)

$\Rightarrow \min tr(-2(Y_c^T X_c R^T)) \Rightarrow \max tr(U \Sigma V^T R^T) \Rightarrow \max tr(Z V^T R^T U)$

($Y_c^T X_c = U \Sigma V^T$; trace)

$\Rightarrow tr(\Sigma)$ already maximal, therefore find such R that cancels V^T and $U \Rightarrow R = VU^T$ so $tr(ZV^T R^T U) = tr(\Sigma)$.

But if $|R| = |VU^T| = -1$ (mirroring) $\Rightarrow R = VSU^T$ $S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$

(to flip sign)

translation: 1. $y_i = Rx_i + t$

2. $R(x_i - \bar{x}) = y_i - \bar{y} \Rightarrow y_i = Rx_i - R\bar{x} + \bar{y}$

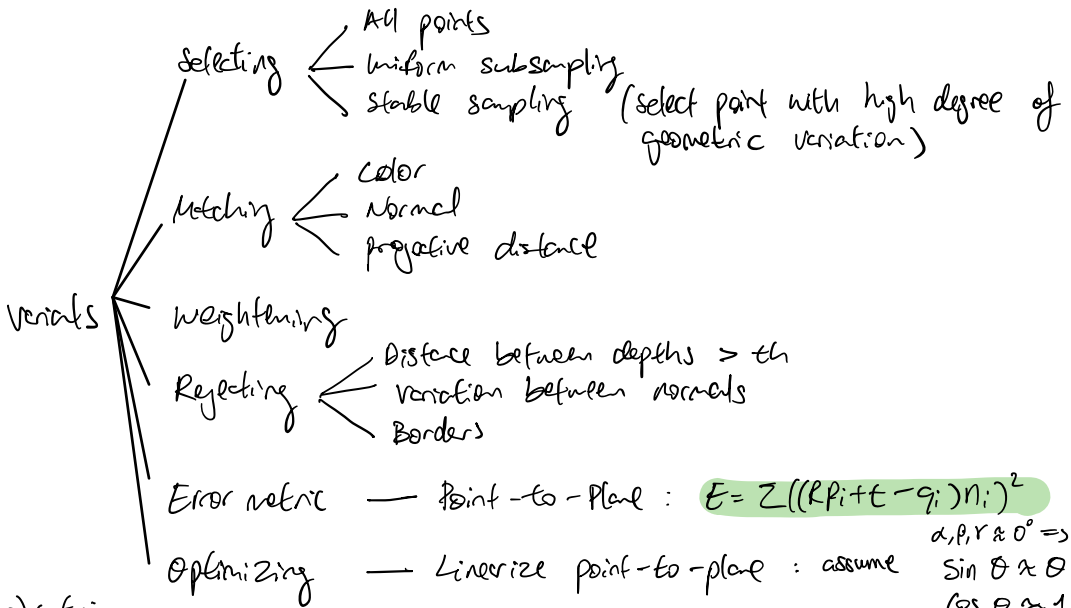
$Rx_i + t = Rx_i - R\bar{x} + \bar{y} \Rightarrow t = \bar{y} - R\bar{x}$

ICP (no correspondence)

Iterative method that at each step approximate correspondence as closest point pairs and find aligned (R, t) based on this. Converge if consequent pose are close enough.

Basic version:

- Select random points
- Match points on the other scan (NN)
- Reject outliers (distance > k · median)
- Minimize $E = \Sigma (Rp_i + t - q_i)^2$



Tricks: coarse-to-fine for faster convergence and avoid local minima

Online 3D Reconstruction

1. Capture depth map
2. Compute pose between all frames
3. Integrate data into model (volumetric fusion)
4. Extract mesh / render SDF (RayCasting)

Volumetric fusion

weighted average of TSDF (from different views / frames)

$$D(x) = \frac{\sum w_i(x) d_i(x)}{W(x)}$$

i = camera

$W(x) = \sum w_i(x)$ normalizing term

$w_i(x)$ based on distance, further away lower weight

Incrementally:

$$D_{i+1}(x) = \frac{W_i(x) D_i(x) + w_{i+1}(x) d_{i+1}(x)}{W_i(x) + w_{i+1}(x)} \qquad W_{i+1}(x) = W_i(x) + w_{i+1}(x)$$

Not true SDF:

Less accurate the further away
More accurate with more views

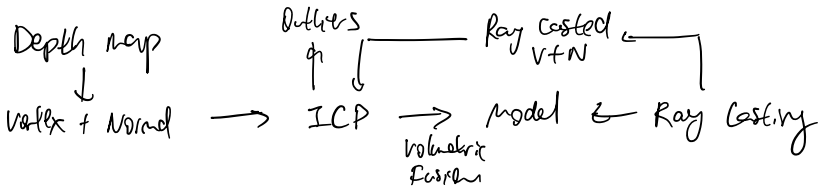
0 -> surface
+ -> free space
- -> unknown

free space carving is important

Ray Casting

To render SDF, cast rays through focal points of each pixel.
Traverse voxels until finding sign change (surface), compute intersection.

Kinect Fusion



Issue: runs out of memory with large TSDF

Extended Kinect Fusion

Rolling Volume instead of whole TSDF. Extract previous parts to mesh.
Problem: cannot revisit previous regions

Hierarchical fusion

Volumes in tree-like structure. Efficient storage, fast lookup (ray casting)
Problem: slow update

Voxel Hashing

Maintain hash array (with position and pointer) and pointed data array (voxels).

Each query (x,y,z) need to be hashed (spatial hash fnc) to find hash entry and then with the corresponding pointer find the voxel.

fast update, but slow lookup (for ray casting)

Point-based

Surfel representation: point + normal + radius. Problem: lower quality than SDF

Loop closure issue

Appears when small errors (drifts) gets accumulated resulting in loop closure problem. Common in local optimizations like ICP of cumulative frames

Elastic Fusion

exploit surfel representation. Explicitly detect loop closure through localization and non-rigidly warp representation to close loop.

Bundle fusion

Cumulative RGB-D bundling:

1. Find correspondese (SIFT)
2. Optimize $E(T) = w_{sparse} E_{sparse}(T) + w_{dense} E_{dense}(T)$

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

$$E_{dense}(T) = w_{depth} E_{depth}(T) + w_{color} E_{color}(T)$$

$$E_{depth} = \sum_{i,j} \sum_{k} \| (p_{ik} - T_i^{-1} T_j \pi_d^{-1}(D_j(\pi_d(T_j^{-1} T_i p_{ik})))) \cdot n_{ik} \|_2^2$$

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

first optimize E_{sparse} and then jointly $E_{sparse} + E_{dense}$

3. Scene (model) update

Solve loop closure by de-integrating frames $D'(v) = \frac{w(v)D(v) - w_n(v)d_n(v)}{w(v) - w_n(v)}$

$w'(v) = w(v) - w_n(v)$

Non-Rigid Deformations

Mesh Deformation

To deform meshes non-rigidly each vertex might have a different transformation.

Handlers (selected vertices) are added to the mesh, so user can drag these to desired locations. E_{fit}

Regularization terms are added to move neighbor vertices (fan) accordingly. E_{reg}.

$E(V,X) = \lambda_{fit} E_{fit} + \lambda_{reg} E_{reg}$. $E_{fit}(V) = \sum_i \|c_i - v_i\|_2^2$ over picked points

Laplacian Surface Editing

Enforce smoothness on deformations. Make neighbors move same way as 1 move.

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

As-Rigid-as-Possible (ARAP)

Each local neighborhood is associated a different rotation (3 DoG).
Thus, each vertex has (3 local rot + 3 trans = 6 DoG)

$E_{reg} = E_{ARAP}(R,V) = \sum_i \sum_{j \in N(i)} \|(v_i - v_j) - R_i(v_i' - v_j')\|_2^2$ Parametrize R_i with Euler angles or quaternion

params $= 3 \cdot n + 3 \cdot n$
constraints $= \underbrace{3 \cdot n \cdot \text{avg}|\text{fan}|}_{E_{reg}} + \underbrace{3 \cdot \#\text{handler}}_{E_{fit}}$

} Alternated optimization (assume one fixed)
vertices (translation) $\leftrightarrow$ rotation $\Rightarrow$ Linear (Procrustes)

could add $w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})$

Embedded Deformation (ED)

More DoG than ARAP by not restricting local transformation matrix to rotations (9 vs 3 DoG).

$E_{reg} = E_{ED}(M,V) = \sum_i \sum_{j \in N(i)} \|(v_i - v_j) - M_i(v_i' - v_j')\|_2^2$. To enforce $M_i \approx R_i$ (orthogonality):

$E_{rot}(M) = \sum_i Rot(M_i)$, $Rot(M_i) = \underbrace{(c_1 \cdot c_2)^2 + (c_2 \cdot c_3)^2 + (c_1 \cdot c_3)^2}_{c_1 \perp c_2 \perp c_3} + \underbrace{(c_1^2 - 1) + (c_2^2 - 1) + (c_3^2 - 1)}_{c_i \cdot c_i = 1}$

Harder to optimize (quartic problem)

Deformation Proxies

Deformation energy on proxies instead of directly on the mesh. Makes the problem sparser (less DoF).

Cage

(Parts of) mesh tightly enclosed into a cage. Same idea as barycentric coordinates, vertex inside cage computed as linear combination of cage vertices.

Cage (C_1, C_2, C_3) : $V_1 = 0.4\,C_1 + 0.1\,C_2 + 0.5\,C_3$

Mesh can be deformed by animating cage.

3D Grid

ARAP deformation on 3D grid for realtime performance. Mesh is fitted into grids (voxels).

Skeleton

Often used in human body reconstruction / tracking. Embed a skeleton inside the mesh, all vertices has some dependency on this joint depending on the part it belongs. Therefore, each joint has a different trasformation M_i.

$$v_i = \sum_u w_{iu} M_u v_i'$$

Deformation Graphs

Applied on ED, each point takes weighted average transformation M_i from k closest graph nodes. n control nodes that affects the neighborhood.

$$p \mapsto \sum_u w_u M_u$$

Non-Rigid Tracking

correspondence between frames $\Rightarrow$ moved handler.
Find correspondence in ICP way or SIFT features.

Non-Rigid ICP

Same as standard ICP but with more DOG (deformation graph). Need some initial alignment

Iteratively: find correspondence (NN / projective)
solve for deformation and vertices

$$E = \lambda_{pair} E_{point} + \lambda_{plane} E_{plane} + \lambda_{color} E_{color} + \lambda_{reg} E_{reg}$$

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

$$E_{plane} = \sum_i \| (v_i - T v_i') n_i \|_2^2$$

$$E_{color} = \sum_i \| I(\pi_i(v_i)) - I'(\pi_i(T v_i)) \|_2^2$$

$$E_{reg} = ARAP = \sum_i \sum_j \| (v_i - v_j) - R_i(v_i' - v_j') \|_2^2$$

} Correspondence

Optimizing for v_i', T, R_i.

Template acquisition

Multi resolution template (coarse-to-fine) with prolongation operators.
Needs a predefined template. Stored with voxel hashing

Correspondence

Pruning: normal, distance, angle...

Association: projective lookup, search in kernel region one block per vertex

Optimizing

Multi resolution Gauss-Newton with inexact PCG steps (linear system).

Exploit sparsity of J^T for maximal performance on GPU.

Multi camera setup

With 8 cameras we get more constraints (correspondence), so less occlusion.

Non-Rigid Tracking and Reconstruction

Reconstruction: similar pipeline like the rigid case, KinectFusion.

Build initial SDF volume from depth map of 1st frame.
In the next frames, estimate deformation from canonical undeformed model to current depth map, warp it and integrate current frame's depth map (volumetric fusion).

For each frame:
1. Take depth map
2. Extract depth map from prev. warped model (Marching cubes + rasterization)
3. Align canonical model (depth map) with current frame's => warped model
4. Integrate current frame into canonical model
5. Next frame

Tracking vs Reconstruction

Tracking => offline template reconstruction + online template tracking.
Mitigate drift with regularizer wrt mesh.

Reconstruction => joint reconstruction and tracking (more DoFs, more difficult).
Template tracking wrt prev. frame => accumulated drift => reset hey volume every n frames.

Challenging: RGB-D sequence under-constraint, need to hallucinate data through regularization (DL).

Dynamic Fusion

ED (Deformation graph) + N-ICP (dense) + volumetric fusion.

DoFs determined by number of deformation graph nodes.

Limitation: in fast motion hard to determine correspondence, is the surface deforming or is new surface being scanned?

Volume Deform

ARAP (3D grid) + N-ICP (dense depth + SIFT) + volumetric fusion

Overcome Dynamic Fusion's limitation with extra SIFT features for correspondence finding

More grid points => more resolution

Real-time Geometry, Albedo and Motion Reconstruction

ED (deformation graph) + N-ICP (dense depth + dense color) + Illumination correction + volumetric fusion

Optimizing the color and illumination helps with the correspondence matching in fast motion.

Fusion 4D

More cameras (8), more complex system. Multiple canonical poses.
Can handle some topology change.

Motion 2 Fusion

Same pipeline as Fusion4D but improved efficiency. 200 FPS

ED (deformation graph) + key volumes + multiple cameras + ML correspondences + two way ICP alignment (current ↔ reference) + Detail layer (2.5D volume)

Looking Good

Same pipeline. Novelty => neural re-rendering (DL)

Deep Deform

Same pipeline. Novelty => DL learned correspondences (between current and canonical model)
↳ heat map
(stacked hourglass)

Body Fusion

Same pipeline but done with skeleton proxy => less DoFs (easier problem).

Face Reconstruction

Done with parametric model (proxy) for dimensionality reduction purpose. Vertices (position) are controlled by parameters that are weights for basis functions.

$P = (\Phi, \alpha, \beta, \delta, \gamma)$

Φ = [R,t] extrinsics of camera
α = params corresponding to shape basis
β = " albedo basis
δ = " expression basis
γ = " sh illumination basis

$M_{geo}(\alpha, \delta) = \mu_{id} + E_{id} \cdot \alpha + E_{exp} \cdot \delta$
$M_{alb}(\beta) = \mu_{alb} + E_{alb} \cdot \beta$
$Illum(\gamma) = \sum_i \gamma_i \beta_i(\vec{n})$

μ = basis mean
E = basis
β_i sh basis function, $\vec{n}$ point normal

Illumination

Often hard (costly) problem in computer grafics => advanced light transport models intractable.

We use simplified models

Environment Map

Reflection map. Panoramic image that works as a texture map for the mesh to bring illumination.

Assumptions: distant lighting, no self shadowing, no scattering

Sphere map

Cube map

Preferred over sphere map as it computationally cheaper, no distortion and not dependent on view point.

Spherical Harmonics

parametrization (approximation of env. map) with orthogonal basis defined over a sphere. With increasing number of bands (order) the approximation is more accurate. Each basis takes the normal of the point and simulates the incident lighting. $L = \sum \gamma_i B_i(\vec{n})$ 3 bands => 9 params (basis)

This illumination model assumes a Lambertian (diffuse) surface, that is lighting is invariant to viewpoint change.

Distant smooth lighting as env. maps.

Parametric Model

Models constructed with a set of different face scans.

At learning time (build database / basis):
- Scan faces / expressions and fit a topologically-consistent template to each with non-rigid alignment
- Compute PCA-basis (set of eigenvectors) for identity, expression, albedo

At test time (fit or construct face from input image):
- find model parameters that fits input RGB(-D) face image, optimization.

Morphable Models

Two independent PCA for identity and albedo.
More general. Parameters have global influence. Mostly caucassian people scan (bias).

Blendshapes

Additive model controlled via blend weight. E.g. first eigenvector (basis) controls face lenght...
More personalized, more semantic meaning for the basis

Principal Component Analysis (PCA)

Used for dimensionality reduction => compute face prior, the basis.

Given $X = \begin{bmatrix} | & | & \\ x_1 & x_2 & \cdots \\ | & | & \end{bmatrix}$, where x_i are face meshes computed with non-rigid deformation over template mesh to align input scans.

1. Compute mean

$$\bar{x} = \frac{1}{N}\sum x_i \qquad X_c = X - \bar{x}$$

2. Eigen decomposition (on covariance $X_c^T X_c$ as it's squared)

$X_c^T X_c B_i = \lambda_i B_i$ with power iteration. $\Rightarrow B_i$ basis, λ_i eigenvalue (variance).

$$E_{id} = \begin{bmatrix} | & | & \\ B_1 & B_2 & \cdots \\ | & | & \end{bmatrix}$$

λ_i ordered decreasingly $\Rightarrow$ leave the ones with low variance. 90% is often enought. Dim. reduction

Could also use SVD instead of PCA

Fitting

Analysis-by-synthesis: aims to analyze a signal or image by reproducing it using a model (find parameters that synthesize the closest image possible in the span of the model).

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

$$E_{sparse}(P) = \sum \| c_i - \pi(M_i(v_j)) \|_2^2 \quad \text{pixel space}$$

$$E_{reg}(P) = \sum \frac{\alpha_i}{\sigma_{id,i}} + \sum \frac{\beta_i}{\sigma_{alb,i}} + \sum \frac{\delta_i}{\sigma_{exp,i}} \quad \text{parameter space. Helps to be } [-30, 30] \text{ to the mean.}$$

Ill-posed problem

$$\text{dense} \begin{cases} E_{geom}(P) = \sum \| \underbrace{M_i(P) - D(\pi(M_i(P)))}_{\text{point-to-point}} \|_2^2 + [\underbrace{(M_i(P) - D(\pi(M_i(P)))) \cdot n_i}_{\text{point-to-plane}}]^2 \\ E_{color}(P) = \sum \| M_i(P)_c - I(\pi(M_i(P))) \|_2^2 \end{cases}$$

over rendered face mask. But in world space

Differencial rendering

Dense terms' residuals are calculated in pixel space, but we want the partial derivatives to affect the vertices/colors in world space.

Each pixel in the rendered image comes from a triangle in world space with location defined by the barycentric coordinate inside the triangle. If we use (save) this information, then we are able to calculate the derivatives.

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

(α, β, γ) barycentric coordinate. The color is just a linear combination of the triangle's vertices' color.

Tricks

The computation cost comes from the dense color terms.
To make it work in real-time we can adopt a coarse-to-fine strategy which not only helps the speed but also the convergence (smoother function). The result from a lower resolution optimization are used as initialization for next level.

RGB Only

Also doable with RGB only by dropping the E_{geo} terms. But need several different frames to jointly optimize for the identity and albedo (base color, shape, texture). Then, these are fixed and only expression, extrinsics and illumination are optimized.

The mesh would not look good, but the rendered image is still ok.

Mouse, Teeth

Use proxies: add generic teeth mesh inside mouth. Depends on RGB-D

An improved solution (RGB only) is to save the sequence cropped mouth + teeth and compute a descriptor for each. Then after the expression tracker, use parameters such expression (δ), landmarks, R and define some similarity metric to find the best matching frame.

Body and Hand

Same approach as faces, parametric model.

- Take a set of scans from variety of subject / poses
- Non-rigidly align them with template for topology consistency.

Body

pose defined from embedded skeleton.

$M(\vec{\theta}, \vec{\beta}, \vec{u}, \Phi)$

$\vec{\theta}$ = pose (skeleton)
$\vec{\beta}$ = shape
$\vec{u}$ = texture
Φ = hyper params

CAESAR

One of the first human body scan database.

125 male + 125 female all neutral pose.
74 markers (landmarks) to help the alignment.

Can't handle pose

SCAPE

Expansion of CAESAR with more identity and poses.
37 identity + CAESA, 70 poses of a particular person.

Parametric (PCA) model for base shape + Rigid (skeleton) and Non-rigid muscle flex.
$P = (R, \beta, Y)$ alternating optimization. Q (corresponding to muscle flex) can be obtained using the given α and optimized R

joint rotation, shape, predicted mesh

SMPL

Novel way to parametrize pose by skeleton. Seperate human body by parts controlled by skeleton joint's movement [R, t].

The rotation can be represented with 3x3 mat then adding constraints to enforce orthogonality ($\approx$ ED) or better with exponential maps ($so(3)$) to reduce dimentionality.

$\|\vec{\omega_j}\|$ angle of rotation $\vec{\omega_j}$ axis of rotation.

$R = e^{\hat{\vec{\omega}}} = I + \hat{\vec{\omega}} \sin(\|\vec{\omega_j}\|) + \hat{\vec{\omega}_j}^2 (1 - \cos(\|\vec{\omega_j}\|))$.

Get back to lie group $SO(3)$ from skew-symmetric $\hat{\vec{\omega}} \in so(3)$

So each joint's pose now is represented by

$$G(\vec{\omega}, j) = \begin{bmatrix} e^{\vec{\omega}}_{3\times3} & j_{3\times1} \\ 0 & 1 \end{bmatrix}$$

The total motion of a vertex (or part) is affected by the concatenation of all joint's movement:

$$P_S = G(\vec{\omega_1}, \vec{\omega_2}, j_1, j_2) = G(\vec{\omega_1}, j_1)\, G(\vec{\omega_2}, j_2)\, \bar{P}_b \quad \Rightarrow$$

$$\bar{t}_i' = \sum_k w_{ki}\, G_k'(\vec{\theta}, J)\, \bar{t}_i$$

k = joints for smooth transition (blended linear comb.)
w_{ki} = weight that a vertex has for a joint k

$M(\vec{\beta}, \vec{\theta}; \Phi) : \mathbb{R}^{|\theta|\times|\beta|} \to \mathbb{R}^{3N}$ the shape model. $\vec{\beta}$: shape params, $\vec{\theta}$: joint rotations
$W(\bar{T}, J, W, \vec{\theta}) : \mathbb{R}^{\cdots} \to \mathbb{R}^{3N}$ gives the displacements of each vertex. transf.
$\bar{T}$ = template mesh, J = joints, W = weights, $\vec{\theta}$ = joint rotation

Training $\Rightarrow$ find $\Phi = (\bar{T}, W, J, S, P)$. Inference $\Rightarrow$ find $\vec{\beta}, \vec{\theta}$
the model. S shape basis
P (joint) pose basis

LBS Problem

Artifacts (unnatural local deformation) from linear blend skinning, as it's just a linear function. $\Rightarrow$ Muscles should flexed as pose changes.

Solution: a blend shape with a set of vertex displacement relative from the neutral pose (offsets)

When the pose is changed, the base shape should also be changed in a small degree (offset)

Reconstruction

Fit template to input scans (SMPL model):

$$E = w_d E_d + w_s E_s + w_m E_m$$

$E_d = \sum w_i \cdot dist^2(T_i v_i, D)$ data term (2D silhouette / 3D)
$E_s = \sum_{(i,j)\in Edge} \|T_i - T_j\|_F^2$ smoothness regularization term
$E_m = \sum \|T_{ki} v_{ki} - m_i\|^2$ marker term (2D)

Possible with RGB only with silhouettes and segmentation...

Hand

Similar idea β = shape θ = pose. However dataset are usually built with synthetic data as scanning hands is usually difficult.

2 approaches
- Discriminative: per-frame reinitialization to prevent error propagation
- Generative: template mesh deforming to align with captured depth data

Taylor et al. 16

Subdivision model which is mostly C_2 continuous (twice differentiable) => fast LM fitting.

Fit kinect input data to sparse points of hand model.

Blend shape + linear blend skinning for pose.

At each step:
1. Preprocess
2. Generate starting point from last frame's prediction (reinitializer)
3. Optimize all $E(\theta)$
4. Take the one that yields lowest cost

$$E(\theta) = E_{data} + \lambda_{bg} E_{bg} + \lambda_{pose} E_{pose} + \lambda_{limit} E_{limit} + \lambda_{tmp} E_{tmp} + \lambda_{int} E_{int} + \lambda_{tips} E_{tips}$$

Sphere-mesh (Tkach)

sphere proxies for the hand => articulated and flexible components.

Hybrid approach: discriminative (estimate the template) + generative (refine results, real-time tracking)

Similar energy formulation than Taylor et. al.

Posterior work: added Kalman filter

Issue of parametric model

Need expensive scanning setup to build database (PCA)
Neutral pose is not identical between subjects
Ambiguity to distinguish identity or pose change (need regularizers)
Drift in non-rigid registration (useless PCs)