Least Squares Approach for Computer Graphics
(From Point Cloud to CAD Models - A Brief Introduction)
Motivation

- From 3D points to CAD models
- Local surface fitting to 3D points
Reverse Engineering

• From physical prototypes to digital prototypes via reverse engineering
2D Terrain Modeling

• A simplified case
Motivation

• Given data points, fit a function that is “close” to the points

\[ y = f(x) \]

\[ P_i = (x_i, y_i) \]
Outline

• Least squares approach
  – General / Polynomial fitting
  – Linear systems of equations
  – Local polynomial surface fitting
Line Fitting

• $y$-offset minimization

\[ P_i = (x_i, y_i) \]
Line Fitting

• Orthogonal offset minimization – Principal Component Analysis (PCA)
Line Fitting

• Find a line $y = ax + b$ that minimizes

$$E(a,b) = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2$$

• $E(a,b)$ is quadratic in the unknown parameters $a, b$

• Another option would be, for example:

$$AbsErr(a,b) = \sum_{i=1}^{n} |y_i - (ax_i + b)|$$

• But – it is not differentiable, harder to minimize...
Line Fitting – LS Minimization

• To find optimal $a, b$ we differentiate $E(a, b)$:

$$
\frac{\partial}{\partial a} E(a, b) = \sum_{i=1}^{n} (-2x_i)[y_i - (ax_i + b)] = 0
$$

$$
\frac{\partial}{\partial b} E(a, b) = \sum_{i=1}^{n} (-2)[y_i - (ax_i + b)] = 0
$$
Line Fitting – LS Minimization

- We obtain two linear equations for $a, b$:

\[
\sum_{i=1}^{n} (-2x_i)[y_i - (ax_i + b)] = 0
\]

\[
\sum_{i=1}^{n} (-2)[y_i - (ax_i + b)] = 0
\]
Line Fitting – LS Minimization

• We obtain two linear equations for \(a, b\):

\[
\sum_{i=1}^{n} [x_i y_i - ax_i^2 - bx_i] = 0
\]

(1)

\[
\sum_{i=1}^{n} [y_i - ax_i - b] = 0
\]

(2)
Line Fitting – LS Minimization

• We obtain two linear equations

\[
\left( \sum_{i=1}^{n} x_i^2 \right) a + \left( \sum_{i=1}^{n} x_i \right) b = \sum_{i=1}^{n} x_i y_i
\]

\[
\left( \sum_{i=1}^{n} x_i \right) a + \left( \sum_{i=1}^{n} 1 \right) b = \sum_{i=1}^{n} y_i
\]
Line Fitting – LS Minimization

- Solve for $a, b$ using (for example) Gauss elimination

- Question: why the solution is the *minimum* for the error function?

\[ E(a, b) = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2 \]
Fitting Polynomials
Fitting Polynomials

• Decide on the degree of the polynomial, $k$
• Want to fit $f(x) = a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$
• Minimize:

$$E(a_0, a_1, \ldots, a_k) = \sum_{i=1}^{n} [y_i - (a_k x_i^k + a_{k-1} x_i^{k-1} + \ldots + a_1 x_i + a_0)]^2$$

$$\frac{\partial}{\partial a_m} E(a_0, \ldots, a_k) = \sum_{i=1}^{n} (-2x^m)[y_i - (a_k x_i^k + a_{k-1} x_i^{k-1} + \ldots + a_0)] = 0$$
Fitting Polynomials

- We obtain a linear system of $k+1$ in $k+1$ variables

$$\begin{pmatrix}
\sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\
\sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2k}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_k
\end{pmatrix}
= 
\begin{pmatrix}
\sum_{i=1}^{n} 1 \cdot y_i \\
\sum_{i=1}^{n} x_i y_i \\
\vdots \\
\sum_{i=1}^{n} x_i^k y_i
\end{pmatrix}$$
General Parametric Fitting

• We can use this approach to fit any function $f(x)$
  – Specified by parameters $a, b, c, ...$
  – The expression $f(x)$ linearly depends on the parameters $a, b, c, ...$
General Parametric Fitting

• Want to fit function $f_{abc...}(x)$ to data points $(x_i, y_i)$
  – Define $E(a,b,c,...) = \sum_{i=1}^{n} [y_i - f_{abc...}(x_i)]^2$
  – Solve the linear system

$$\frac{\partial}{\partial a} E(a,b,c,...) = \sum_{i=1}^{n} (-2 \frac{\partial}{\partial a} f_{abc...}(x_i))[y_i - f(x_i)] = 0$$

$$\frac{\partial}{\partial b} E(a,b,c,...) = \sum_{i=1}^{n} (-2 \frac{\partial}{\partial b} f_{abc...}(x_i))[y_i - f(x_i)] = 0$$

$$\vdots$$
General Parametric Fitting

• It can even be some crazy function like

\[
f(x) = \lambda_1 \sin^2 x + \lambda_2 e^{-\frac{x^2}{\sqrt{2\pi}}} + \lambda_3 x^{17}
\]

• Or in general:

\[
f_{\lambda_1, \lambda_1, \ldots, \lambda_k}(x) = \lambda_1 f_1(x) + \lambda_2 f_2(x) + \ldots + \lambda_k f_k(x)
\]
Solving Linear Systems in LS Sense

• Let’s look at the problem a little differently:
  – We have data points \((x_i, y_i)\)
  – We want the function \(f(x)\) to go through the points:
    \[
    \forall \ i = 1, \ldots, n: \quad y_i = f(x_i)
    \]
  – Strict interpolation is in general not possible
    • In polynomials: \(n+1\) points define a unique interpolation polynomial of degree \(n\).
    • So, if we have 1000 points and want a cubic polynomial, we probably won’t find it...
Solving Linear Systems in LS Sense

• We have an over-determined linear system $n \times k$:

\[
\begin{align*}
  f(x_1) &= \lambda_1 f_1(x_1) + \lambda_2 f_2(x_1) + \ldots + \lambda_k f_k(x_1) = y_1 \\
  f(x_2) &= \lambda_1 f_1(x_2) + \lambda_2 f_2(x_2) + \ldots + \lambda_k f_k(x_2) = y_2 \\
  & \quad \vdots \\
  f(x_n) &= \lambda_1 f_1(x_n) + \lambda_2 f_2(x_n) + \ldots + \lambda_k f_k(x_n) = y_n
\end{align*}
\]
Solving Linear Systems in LS Sense

- In matrix form:

\[
\begin{pmatrix}
  f_1(x_1) & f_2(x_1) & \ldots & f_k(x_1) \\
  f_1(x_2) & f_2(x_2) & \ldots & f_k(x_2) \\
  \vdots & \vdots & \ddots & \vdots \\
  f_1(x_n) & f_2(x_n) & \ldots & f_k(x_n)
\end{pmatrix}
\begin{pmatrix}
  \lambda_1 \\
  \lambda_2 \\
  \vdots \\
  \lambda_k
\end{pmatrix}
= 
\begin{pmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{pmatrix}
\]
Solving Linear Systems in LS Sense

• In matrix form:

\[ A\mathbf{v} = \mathbf{y} \]

where \( A = \left( f_j(x_i) \right)_{i,j} \) is a rectangular \( n \times k \) matrix, \( n > k \)

\[ \mathbf{v} = (\lambda_1, \lambda_2, \ldots, \lambda_k)^T \]

\[ \mathbf{y} = (y_1, y_2, \ldots, y_n)^T \]
Solving Linear Systems in LS Sense

• More constraints than variables – no exact solutions generally exist
• We want to find something that is an “approximate solution”:

$$\tilde{v} = \arg \min_v \| A v - y \|^2$$
Finding the LS Solution

- $\mathbf{v} \in \mathbb{R}^k$
- $A\mathbf{v} \in \mathbb{R}^n$
- As we vary $\mathbf{v}$, $A\mathbf{v}$ varies over the linear subspace of $\mathbb{R}^n$ spanned by the columns of $A$:

$$A\mathbf{v} = \begin{pmatrix} A_1 & A_2 & \cdots & A_k \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_k \end{pmatrix} = \lambda_1 A_1 + \lambda_2 A_2 + \cdots + \lambda_k A_k$$
Finding the LS Solution

- We want to find the closest $Av$ to $y$: $\min_v \|Av - y\|^2$
Finding the LS Solution

• The vector $A\mathbf{v}$ closest to $\mathbf{y}$ satisfies:

$$(A\mathbf{v} - \mathbf{y}) \perp \{\text{subspace of } A\text{'s columns}\}$$

$\forall$ column $A_i$, $\langle A_i, A\mathbf{v} - \mathbf{y} \rangle = 0$

$\forall i, A_i^T (A\mathbf{v} - \mathbf{y}) = 0$

$A^T (A\mathbf{v} - \mathbf{y}) = 0$

$$(A^T A)\mathbf{v} = A^T \mathbf{y}$$

These are called the normal equations
Finding the LS Solution

• We got a square symmetric system \((A^T A) v = A^T y\) (k×k)
• If \(A\) has full rank (the columns of \(A\) are linearly independent) then \((A^T A)\) is invertible.

\[
\min_v \| A v - y \|^2
\]
\[
\downarrow
\]
\[
v = (A^T A)^{-1} A^T y
\]
Weighted Least Squares

• Sometimes the problem also has weights to the constraints:

\[
\min_{\lambda_1, \lambda_2, \ldots, \lambda_k} \sum_{i=1}^{n} w_i [y_i - f_{\lambda_1, \lambda_2, \ldots, \lambda_k}(x_i)]^2, \ w_i > 0 \quad \text{and doesn't depend on } \lambda_i
\]

\[
\Downarrow
\]

\[
\min_v (Av - y)^T W (Av - y), \quad \text{where } W_{ii} = w_i \quad \text{is a diagonal matrix}
\]

\[
\Downarrow
\]

\[
(A^T WA)v = A^T Wy \quad \text{this is a square system}
\]
Motivation

• We are acquiring point cloud directly from scanners
• From physical prototypes to digital prototypes Local surface fitting to 3D points (Reverse Engineering)
Local Surface Fitting to 3D points

• Normals?
• Lighting?
• Upsampling?
Local Surface Fitting to 3D points

Locally approximate a polynomial surface from points
Fitting Local Polynomial

- Fit a local polynomial around a point $P$
Fitting Local Polynomial Surface

- Compute a reference plane that fits the points close to $P$
- Use the local basis defined by the normal to the plane!
Fitting Local Polynomial Surface

- Fit polynomial $z = p(x,y) = ax^2 + bxy + cy^2 + dx + ey + f$
Fitting Local Polynomial Surface

- Fit polynomial \( z = p(x,y) = ax^2 + bxy + cy^2 + dx + ey + f \)
Fitting Local Polynomial Surface

- Fit polynomial $z = p(x,y) = ax^2 + bxy + cy^2 + dx + ey + f$
Fitting Local Polynomial Surface

• Again, solve the system in LS sense:

\[
ax_1^2 + bx_1y_1 + cy_1^2 + dx_1 + ey_1 + f = z_1
\]
\[
ax_2^2 + bx_2y_2 + cy_2^2 + dx_2 + ey_2 + f = z_1
\]
\[\ldots\]
\[
ax_n^2 + bx_ny_n + cy_n^2 + dx_n + ey_n + f = z_n
\]

• Minimize \( \sum \|z_i - p(x_i, y_i)\|^2 \)
Fitting Local Polynomial Surface

• Also possible (and better) to add weights:

\[ \sum w_i \| z_i - p(x_i, y_i) \|^2, \quad w_i > 0 \]

• The weights get smaller as the distance from the origin point grows.