The Parametric Line Equation: 
Fundamental Analysis of 3D Objects

Mike Bailey
mjb@cs.oregonstate.edu

The Line Equation: 
Why “y=mx+b” Isn’t Good Enough for Graphics and Simulation Applications

1. Cannot represent vertical lines (m = \infty)
2. Can only represent infinite lines, not finite line segments
3. Can only represent 2D lines, not 3D

Note: “z = mx + ny + b” is a plane, not a 3D line
Parametric Line Equation: The “Shooting” Form

\[ x = X_0 + t(X_1 - X_0) \]
\[ y = Y_0 + t(Y_1 - Y_0) \]
\[ z = Z_0 + t(Z_1 - Z_0) \]

\[ 0 \leq t \leq 1. \]

Parametric Line Equation: The “Blending” Form

\[ x = (1-t)X_0 + tX_1 \]
\[ y = (1-t)Y_0 + tY_1 \]
\[ z = (1-t)Z_0 + tZ_1 \]

\[ 0 \leq t \leq 1. \]
Example Use: Intersection of two 2D Line Segments

\[
\begin{align*}
  x_{01} &= x_{23} \\
  y_{01} &= y_{23}
\end{align*}
\]

Since, at the point of intersection, \( x_{01} = x_{23} \) and \( y_{01} = y_{23} \)

we have 2 equations in 2 unknowns \((t, u)\)

Solve for \( t^* \) and \( u^* \). If they are not each between 0. and 1., then the infinite lines intersect, but not the finite line segments. Plug \( t^* \) and \( u^* \) back into the equations above to find \((x^*, y^*)\), the point of intersection.

What if \( t^* \) or \( u^* \) is < 0. or > 1. ?

\[
\begin{align*}
  0. \leq t \leq 1. \\
  u < 0.
\end{align*}
\]

It means you have a situation like one of these. The infinite lines intersect, but not the finite line segments.
An Example with Numbers

\[ x_{01} = x_0 + t(x_1 - x_0) \]
\[ y_{01} = y_0 + t(y_1 - y_0) \]
\[ x_{23} = x_2 + u(x_3 - x_2) \]
\[ y_{23} = y_2 + u(y_3 - y_2) \]

\[ x_{01} = 0 + t(3 - 0) \]
\[ y_{01} = 0 + t(3 - 0) \]  
\[ x_{23} = 0 + u(6 - 0) \]
\[ y_{23} = 2 + u(0 - 2) \]

\[ 3t = 6u \]
\[ 3t = 2 - 2u \]

\[ t^* = \frac{1}{2}; u^* = \frac{1}{4} \]
\[ x^* = 1.5; y^* = 1.5 \]

What if the Lines are Parallel?

Well, technically they do intersect, but at infinity. This shows up in the math by the expression for \( t^* \) and \( u^* \) becoming infinitely large, that is, there would be a divide by zero.

So, you need to check the denominator before doing the division.

Since \( t^* \) and \( u^* \) would both be infinity in this case, which is most certainly > 1, that means that the line segments do not intersect.
Linear Blending Shows Up in a Lot of Computer Graphics-Related Applications

You can linearly blend any two quantities with:

\[ Q = (1 - t)Q_0 + tQ_1 \]

color, shape, location, angle, scale factors, •••

Bilinear Interpolation in a Quadrilateral

You can bilinearly blend any 2D quantities within a quadrilateral by writing the line blending equation twice and then blending the two lines with:

\[ Q_B = (1 - t)Q_{01} + tQ_{11} \]

\[ Q_A = (1 - t)Q_{00} + tQ_{10} \]

\[ Q_{AB} = (1 - u)Q_A + uQ_B = (1 - t)(1 - u)Q_{00} + t(1 - u)Q_{10} + (1 - t)uQ_{01} + tuQ_{11} \]
Bilinear Interpolation in a Quadrilateral

You can derive this equation by doing a blend-of-a-blend, like was shown on the last slide, or you can just look at how the terms have to become 0. or 1. whenever \( t = 0. \) or 1. and \( u = 0. \) or 1.

\[
Q_0 = (1-t)Q_{01} + tQ_{11}
\]

For example, right here \( t = 1. \) and \( u = 0. \)
We know that at this location, the coefficient of \( Q_{10} \) has to be 1. and the coefficients of \( Q_{00}, Q_{01}, \) and \( Q_{11} \) have to all be 0.

\[
Q_{AB} = (1-u)Q_0 + uQ_0 = (1-t)(1-u)Q_{00} + t(1-u)Q_{10} + (1-t)uQ_{01} + tuQ_{11}
\]

So, you can actually just write the bilinear equation "by inspection".

Bilinear Interpolation in a Quadrilateral

Drawing a quadrilateral as two triangles: Drawing a quadrilateral with bilinear interpolation:

This diagonal should have some blue and green in it, but it doesn’t

This should be a continuous surface, without a crease
Higher-Order Interpolation

While linear interpolation is the most common type used in graphics and geometric applications, there is no reason we cannot use higher-order interpolants.

**Second Order:**

\[ P(t) = A + Bt + Ct^2 \]

\[
A = P_0
\]

\[
B = -3P_0 + 4P_1 - P_2
\]

\[
C = 2P_0 - 4P_1 + 2P_2
\]

**Third Order:**

\[ P(t) = A + Bt + Ct^2 + Dt^3 \]

(We are going to discuss this more when we get to keyframe animation)