**Engineering, Test & Technology**
Boeing Research & Technology

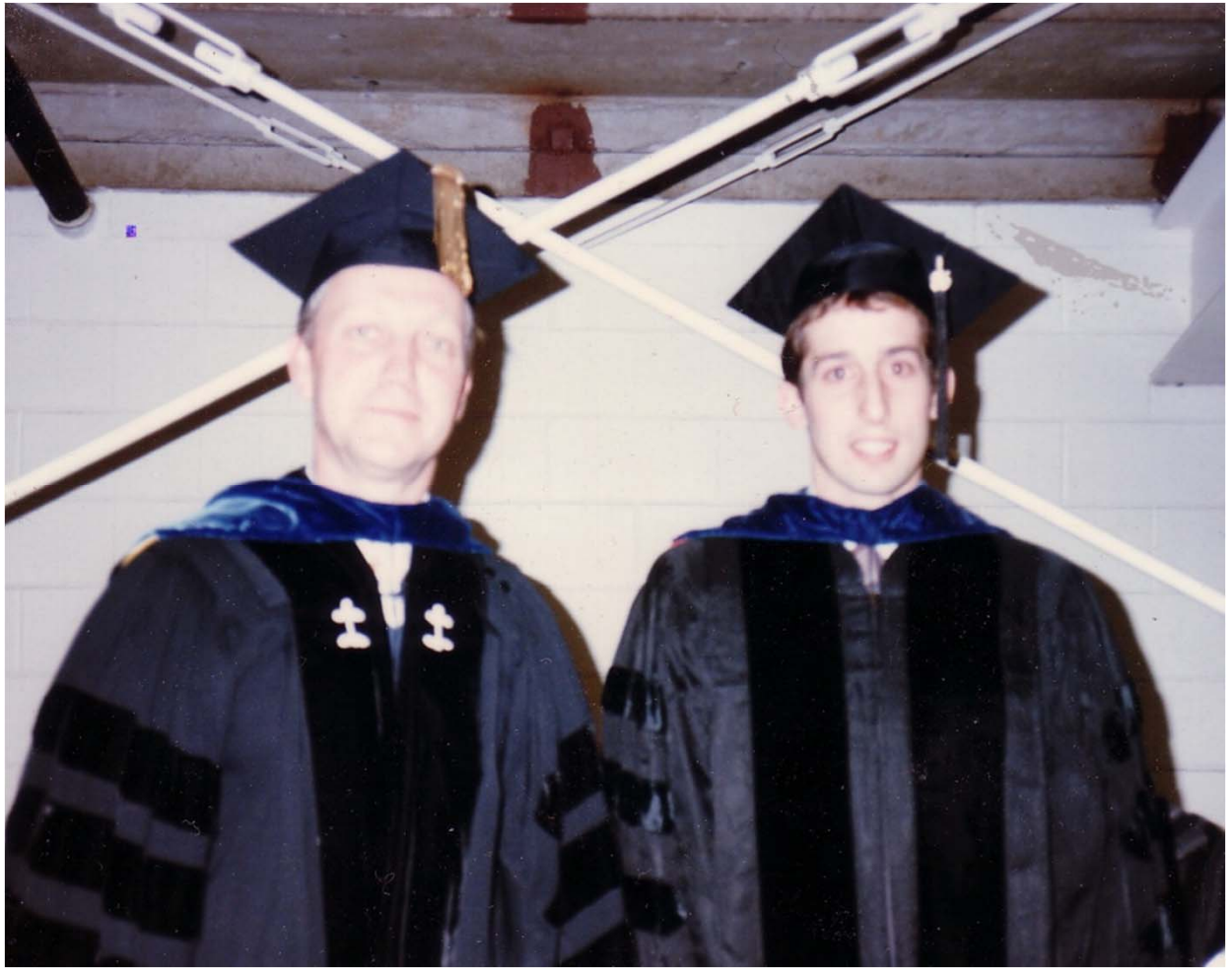
# Solving Systems of Spline Equations: A Linear Programming-based Approach

Thomas Grandine

Senior Technical Fellow

Support and Analytics Technology

# A fun occasion in 1985

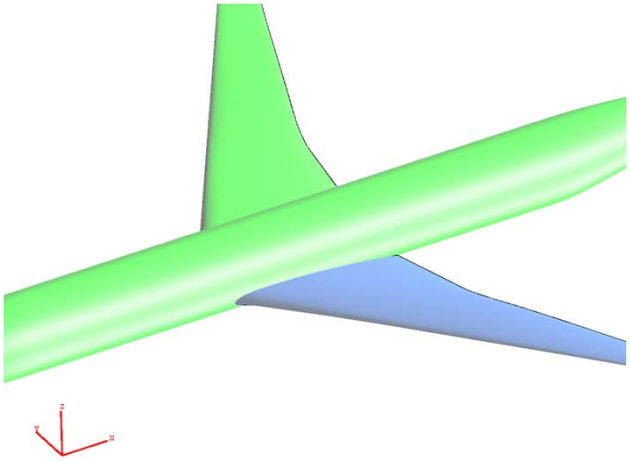


# A few moments later . . .

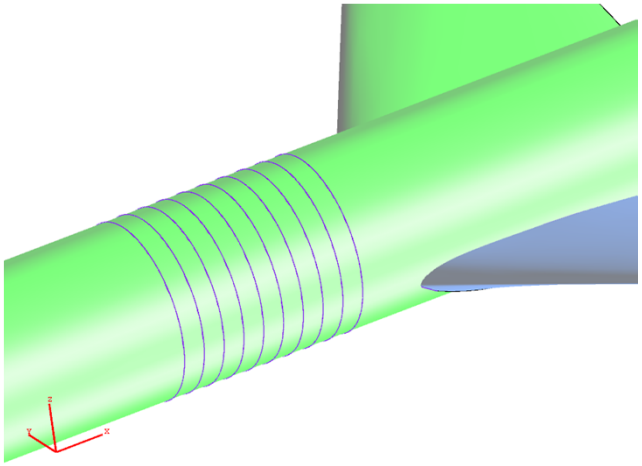
# . . . and 11+ years later in 1997

# Boeing needs to solve approximately 200 million spline systems every day

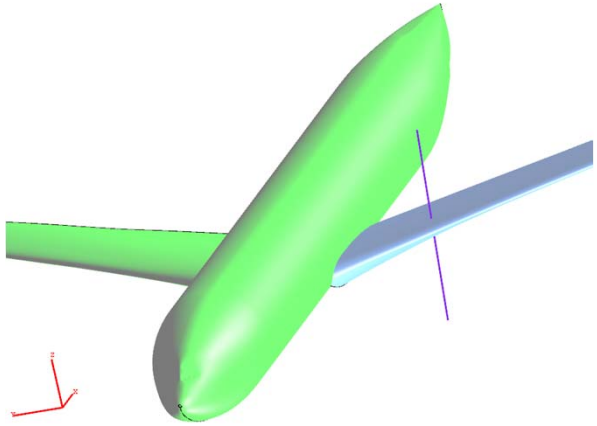
## Applications include



Surface intersection



Planar cut, structured meshing



Ray casting, curve / surface intersection

# Reliability is essential

- **The surface intersection algorithm requires solving, on average, about 20 systems.**
- **If the spline system solver is has 99% reliability, then the surface intersection algorithm is less than 82% reliable.**
- **Even at 99.9% reliability, the surface intersector will still fail 2% of the time.**

# Three references stand out


- [H14] Hanniel, I. “Solving multivariate polynomial systems using hyperplane arithmetic and linear programming,” *Computer-Aided Design* **46**, pp. 101–109.
- [MPR03] Mourrain, B., V. Y. Pan, and O. Ruatta. “Accelerated solution of multivariate polynomial systems of equations,” *SIAM Journal of Computing* **32**, pp. 435–454.
- [SP93] Sherbrooke, E. C. and N. M. Patrikalakis. “Computation of the solutions of nonlinear polynomial systems,” *Computer Aided Geometric Design* **10**, pp. 379–405.


- A few remarks:
  - The papers all exclusively focus on polynomial, rather than spline systems.
  - They eschew algebraic techniques in favor of numerical methods.
  - They focus heavily on robustness.

# The best method for almost 25 years is due to Sherbrooke and Patrikalakis (1993)

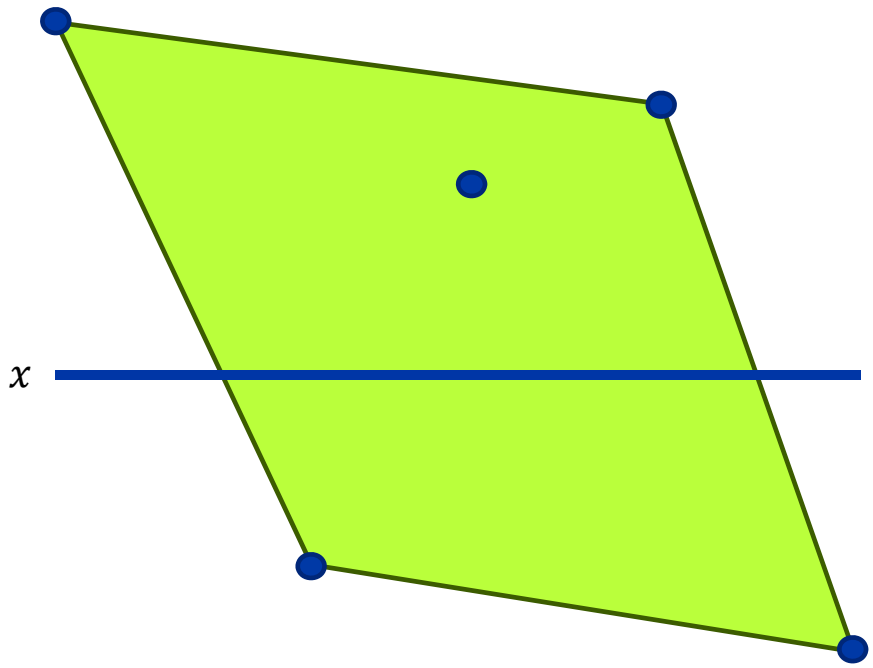
- **Method is usually referred to as The Projected Polyhedron Method**
- **Given a spline function of one variable:**

$$f(x) = \sum_{i=0}^{n} \alpha_i B_i(x) = 0$$

- **Consider the B-spline series for the graph of $f$:**

$$\begin{pmatrix} x \\ f(x) \end{pmatrix} = \sum_{i=0}^{n} \begin{pmatrix} x_i \\ \alpha_i \end{pmatrix} B_i(x)$$

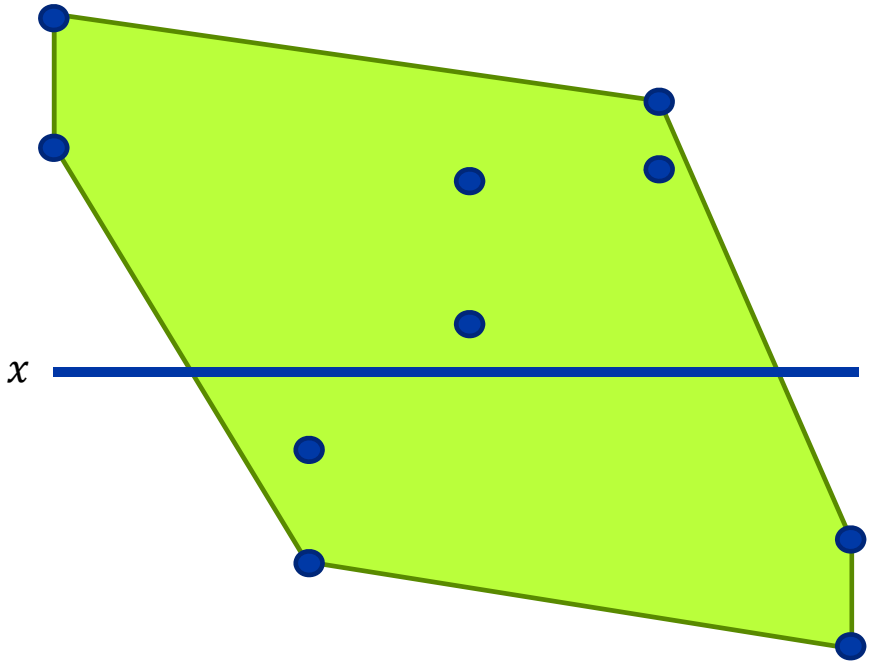
- **The graph of the function lies inside the convex hull of these control points.**



# The projected polyhedron idea generalizes to more than one variable

- **Consider a tensor product spline function:**

$$f(x,y) = \sum_i \left( \sum_j \alpha_{ij} C_j(y) \right) B_i(x) = 0$$




- **This can be thought of as a univariate spline whose B-spline coefficients are functions of the other variables.**
- **For each $i$, consider the two control points**

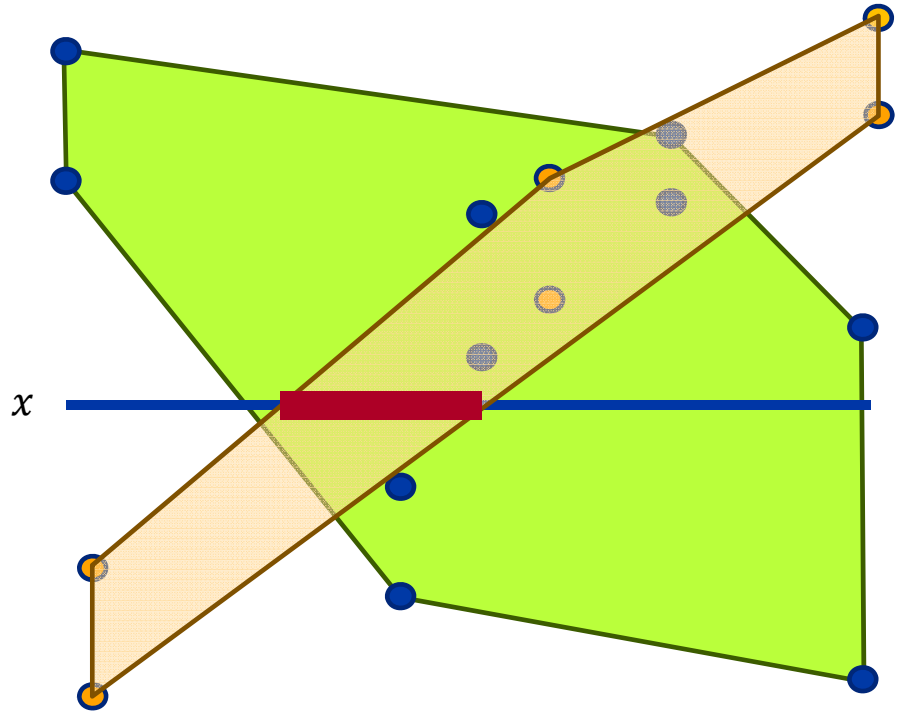
$$\left( \begin{matrix} x_i \\ \min_j \alpha_{ij} \end{matrix} \right) \quad \left( \begin{matrix} x_i \\ \max_j \alpha_{ij} \end{matrix} \right)$$

- **The convex hull trick still works**

# The algorithm considers variables one at a time

- **Suppose one wants to solve**

$$f(x, y) = 0$$
$$g(x, y) = 0$$



# With this building block in hand, the rest of the algorithm is straightforward

1. **Start with first independent variable**
2. **Project onto current independent variable**
3. **Trim splines to smaller interval of definition**
4. **If sufficient amount trimmed from interval (I use 30%):**
    1. Advance to next independent variable; go to step 2.
5. **Otherwise:**
    1. Split domain into two pieces in current independent variable.  Solve recursively.

**Notes:**

- **Algorithm converges linearly**
- **Exhibits all the usual problems with multiple roots**
- **Accuracy can be improved with scaling and shifting**

# Iddo Hanniel has suggested a sharper means of pruning the intervals

- **Considering again the graph of a univariate spline function and its zeros**

$$\begin{pmatrix} x \\ f(x) \end{pmatrix} = \sum_{i=0}^{n} \begin{pmatrix} x_i \\ \alpha_i \end{pmatrix} B_i(x) = \begin{pmatrix} x \\ 0 \end{pmatrix}$$

- **A necessary condition for solving it is that there exist $\lambda_i$ such that**

$$\sum_{0}^{n} \begin{pmatrix} x_i \\ \alpha_i \\ 1 \end{pmatrix} \lambda_i = \begin{pmatrix} x \\ 0 \\ 1 \end{pmatrix}, \qquad \lambda_i \geq 0$$

- **Attaching the objective functions and solving linear programming problems**

$$\min x \qquad \max x$$

- **Creates sharper bounds than are obtained by projected polyhedron**

# Spline systems can be solved using the same methodology

## To solve

$$f(x,y) = \sum_i \alpha_i B_i(x,y) = 0$$
$$g(x,y) = \sum_j \beta_j C_j(x,y) = 0$$

## Refine using the linear programs

$$\min x \quad \max x \quad \min y \quad \max y$$
subject to

$$\sum_i \begin{pmatrix} x_i \\ y_i \\ \alpha_i \\ 1 \end{pmatrix} \lambda_i = \sum_j \begin{pmatrix} x_j \\ y_j \\ \beta_j \\ 1 \end{pmatrix} \mu_i = \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}, \qquad \lambda_i \geq 0, \mu_j \geq 0$$

# Hanniel has dubbed this method the “naïve LP” method

1. **The number of problem variables grows exponentially with dimension**
2. **The computational cost of solving LPs is potentially exponential rather than $\boldsymbol{O(n \log n)}$.**

# Many applications have separable problem structure

- **Consider the curve intersection problem**

$$\boldsymbol{p(u) - q(v) = 0.}$$

- **The constraints for this problem are of the form**

$$\sum_i \sum_j \begin{pmatrix} \boldsymbol{u_i} \\ \boldsymbol{v_j} \\ \boldsymbol{p_i - q_j} \\ \boldsymbol{1} \end{pmatrix} \boldsymbol{\lambda_{ij}} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{0} \\ \boldsymbol{1} \end{pmatrix}.$$

- **These reduce to**

$$\sum_i \begin{pmatrix} \boldsymbol{u_i} \\ \boldsymbol{0} \\ \boldsymbol{p_i} \\ \boldsymbol{1} \\ \boldsymbol{0} \end{pmatrix} \boldsymbol{\lambda_i} + \sum_j \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{v_j} \\ \boldsymbol{-q_j} \\ \boldsymbol{0} \\ \boldsymbol{1} \end{pmatrix} \boldsymbol{\mu_j} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{0} \\ \boldsymbol{1} \\ \boldsymbol{1} \end{pmatrix}.$$

# Even problems with partial separability are amenable to this idea

- **Consider**

$$\boldsymbol{f(u,v,w) = f_1(u,v) + f_2(v,w) = 0.}$$

- **Then**

$$\sum_i \sum_j \boldsymbol{\alpha_{ij} B_{ij}(u,v)} + \sum_j \sum_k \boldsymbol{\beta_{jk} C_{jk}(v,w)} = \boldsymbol{0}$$

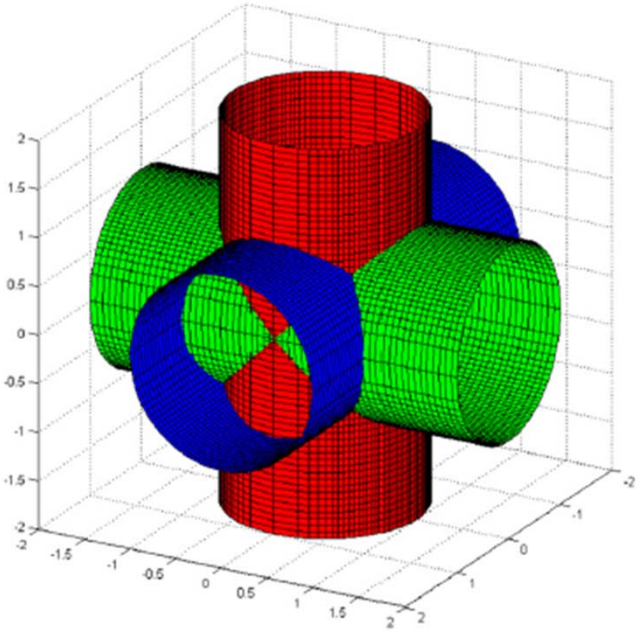
- **Reduces to**

$$\sum_{ij} \begin{pmatrix} \boldsymbol{u_{ij}} \\ \boldsymbol{v_{ij}} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{\alpha_{ij}} \\ \boldsymbol{1} \\ \boldsymbol{0} \end{pmatrix} \boldsymbol{\lambda_{ij}} + \sum_{jk} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{v_{jk}} \\ \boldsymbol{w_{jk}} \\ \boldsymbol{\beta_{jk}} \\ \boldsymbol{0} \\ \boldsymbol{1} \end{pmatrix} \boldsymbol{\mu_{jk}} = \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{0} \\ \boldsymbol{1} \\ \boldsymbol{1} \end{pmatrix}.$$

# What happens when the method is tried on test problems?

- **Consider the test problem**

$$f_i(x_1, \ldots, x_n) = \sum_{j \neq i} x_j^2 - 1 = 0 .$$

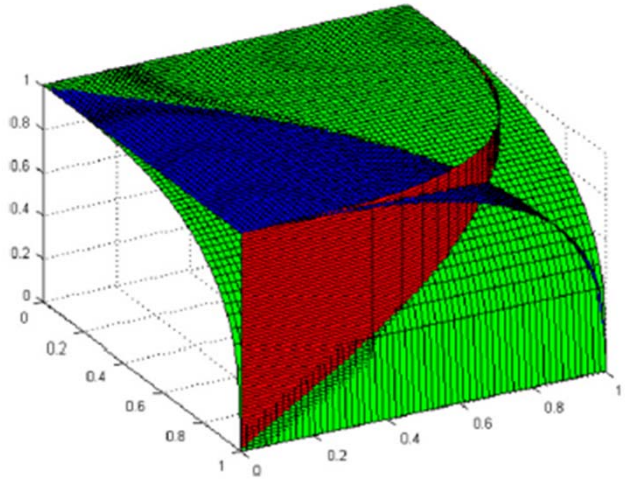


| $n$ | [H14] runtime (ms) | # of LPs | # of solutions |
|---|---|---|---|
| 2 | 12.29 | 13 | 4 |
| 3 | 69.74 | 36 | 8 |
| 4 | 720.5 | 64 | 16 |
| 5 | 6360 | 129 | 32 |
| 6 | 137130 | 257 | 64 |
| 7 | | 513 | 128 |
| 8 | | 1025 | 256 |
| 9 | | 2047 | 512 |

# What happens when the method is tried on test problems?

- **Consider the test problem**

$$f_i(x_1, \ldots, x_n) = \sum_{j \neq i} x_j^3 - 1 = 0\,.$$



| $n$ | [H14] runtime (ms) | # of LPs | # of solutions |
|---|---|---|---|
| 2 | 18.78 | 3 | 1 |
| 3 | 87.4 | 5 | 1 |
| 4 | 1257 | 5 | 1 |
| 5 | 31730 | 5 | 1 |
| 6 | 759257 | 6 | 1 |
| 7 | | 6 | 1 |
| 8 | | 6 | 1 |
| 20 | | 7 | 1 |

# Consider the Broyden tridiagonal system

$$x_{i-1} - (3 - 2x_i)x_i + 2x_{i+1} - 1 = 0, \qquad i = 1, \ldots, n$$

| $n$ | [H14] runtime (ms) | # of LPs | # of solutions |
|---|---|---|---|
| 2 | 19.97 | 11 | 2 |
| 3 | 93.846 | 14 | 2 |
| 4 | 340.81 | 16 | 2 |
| 5 | 1409.1 | 23 | 2 |
| 6 | 6588 | 32 | 2 |
| 7 | | 40 | 2 |
| 8 | | 54 | 2 |
| 20 | | 344 | 2 |

# Things do go bad when the system gets dense

$$\sum_{i=1}^{k} x_i^2 - x_i - \frac{1}{4}\left(\sum_{i=1}^{k} x_i\right)^2 = 0, \qquad k = 1, \ldots, n$$

| $n$ | [H14] runtime (ms) | # of LPs | # of solutions |
|---|---|---|---|
| 2 | na | 7 | 1 |
| 3 | na | 11 | 1 |
| 4 | na | 12 | 1 |
| 5 | na | 13 | 1 |
| 6 | na | 13 | 1 |
| 7 | | 14 | 1 |
| 8 | | ? | |
| 20 | | ? | |

# On one Boeing test problem, results were dramatic

- **Problem was a three surface intersection problem given by 6 equations in 6 unknowns**

$$F(u,v) = G(s,t) = H(a,b).$$

- **Projected polyhedron required over 30000 subdivision steps for this problem.**
- **Linear programming method isolated and found both solutions in 13 subdivision steps.**

# For “naïve LP” to be viable, what is needed?

1. **Much smaller subdivision trees**
2. **A fast LP solver, preferably using sparse linear algebra**
3. **Hot start capability for the LP solver**
4. **Most (all?) workhouse applications to be separable?**
    1. Curve / surface intersection is separable
    2. Surface / surface intersection is separable
    3. Curve projection is separable
    4. Ray tracing is separable
    5. Composition of spline systems are separable

# How does this work for composition of functions?

- **Consider curve on surface / surface intersection:**

$$\boldsymbol{F\big(u(t), v(t)\big) = G(r, s)}$$

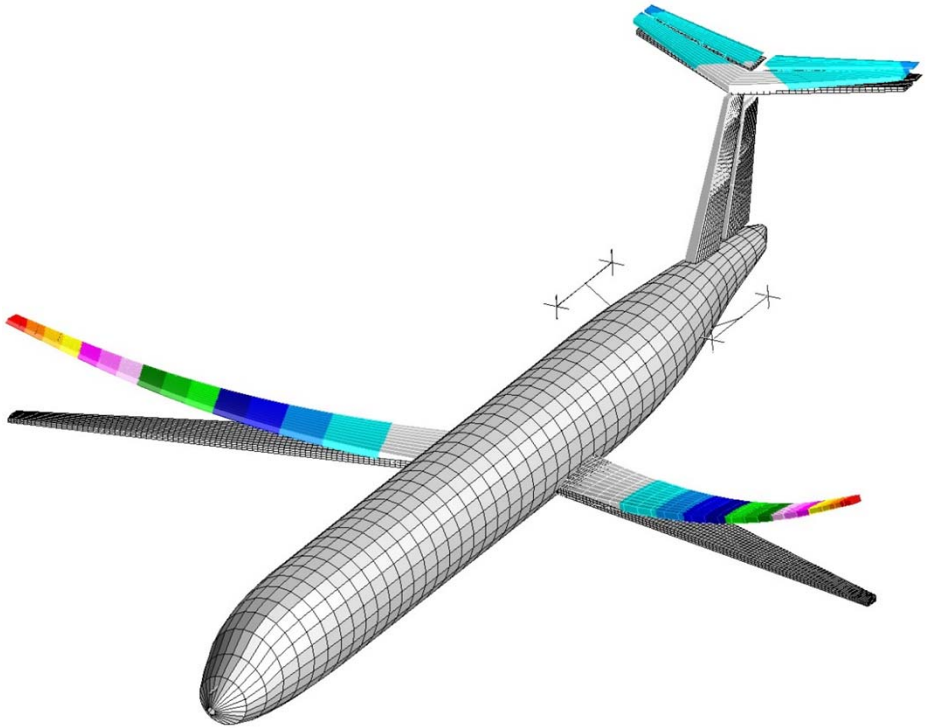
- **This can be reduced to a larger system of the form:**

$$\boldsymbol{F(\hat{u}, \hat{v}) - G(r, s) = 0}$$
$$\boldsymbol{u(t) - \hat{u} = 0}$$
$$\boldsymbol{v(t) - \hat{v} = 0}$$

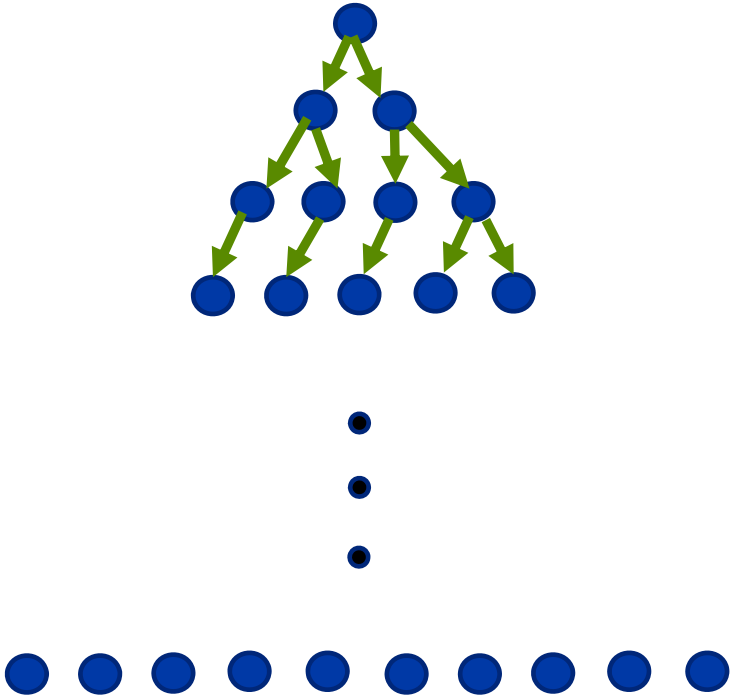
- **This is a $5 \times 5$ system of equations instead of $3 \times 3$, but it is beautifully separable.**
- **This is also a great example of the Betts' tradeoff (trading away nonlinearity for a small increase in problem size).**

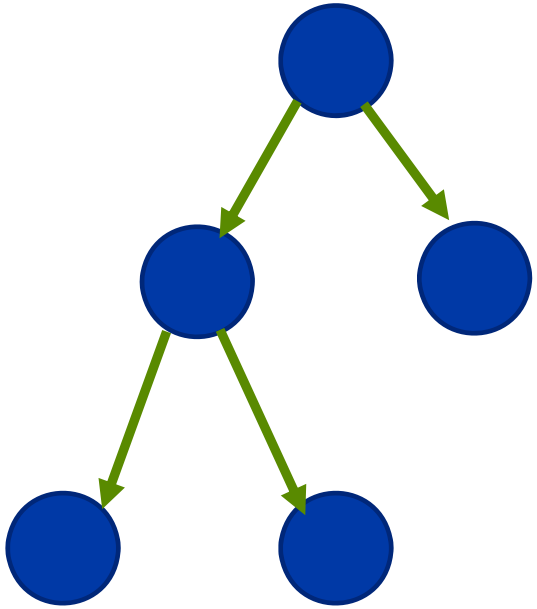
# Many important applications involve composition of functions

- **These include**
  - Structured meshing (by constructing a mesh map so that isoparms $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{s},\boldsymbol{t}),\boldsymbol{v}(\boldsymbol{s},\boldsymbol{t}))$ in $s$ and $t$ form a structured mesh.
  - Deformable solid modeling operations
  - Aeroelasticity
  - Many, many others



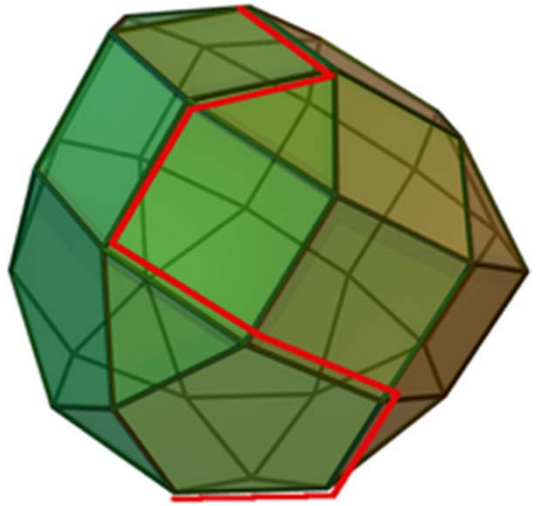
# The method offers considerable potential for parallel processing





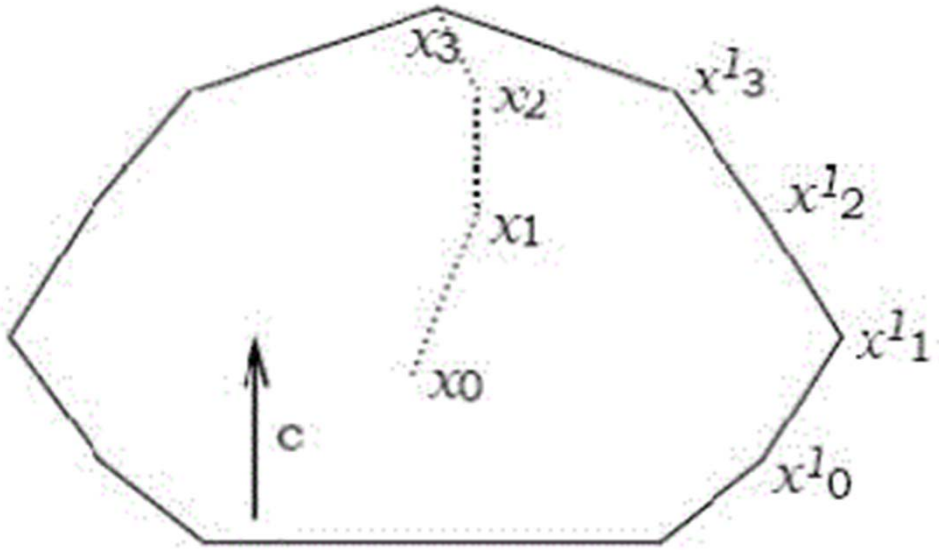
# Implementation details matter in terms of algorithmic performance

- **The simplex method has two phases**
- **The first phase finds a feasible point. Because each of the $2n$ linear programming problems has the same feasible set, it only needs to be performed once.**
- **Because all $2n$ linear programming problems are independent, they may be solved in parallel.**



# The interior point method may be even more advantageous

**The linear system solve at each iteration can be done with $2n$ right-hand-sides, so all problems can be solved simultaneously.**



# Hanniel's "naïve LP" method is much better than advertised

- **Pros:**
  - Scales beautifully on separable and partially separable problems
  - Produces sharp bounds quickly, greatly reducing number of nodes in search tree
  - Offers great potential for parallel implementation
  - Works for more general functions than PP method

- **Cons:**
  - Suffers curse of dimensionality on inseparable problems
  - Slower than PP method on easy problems