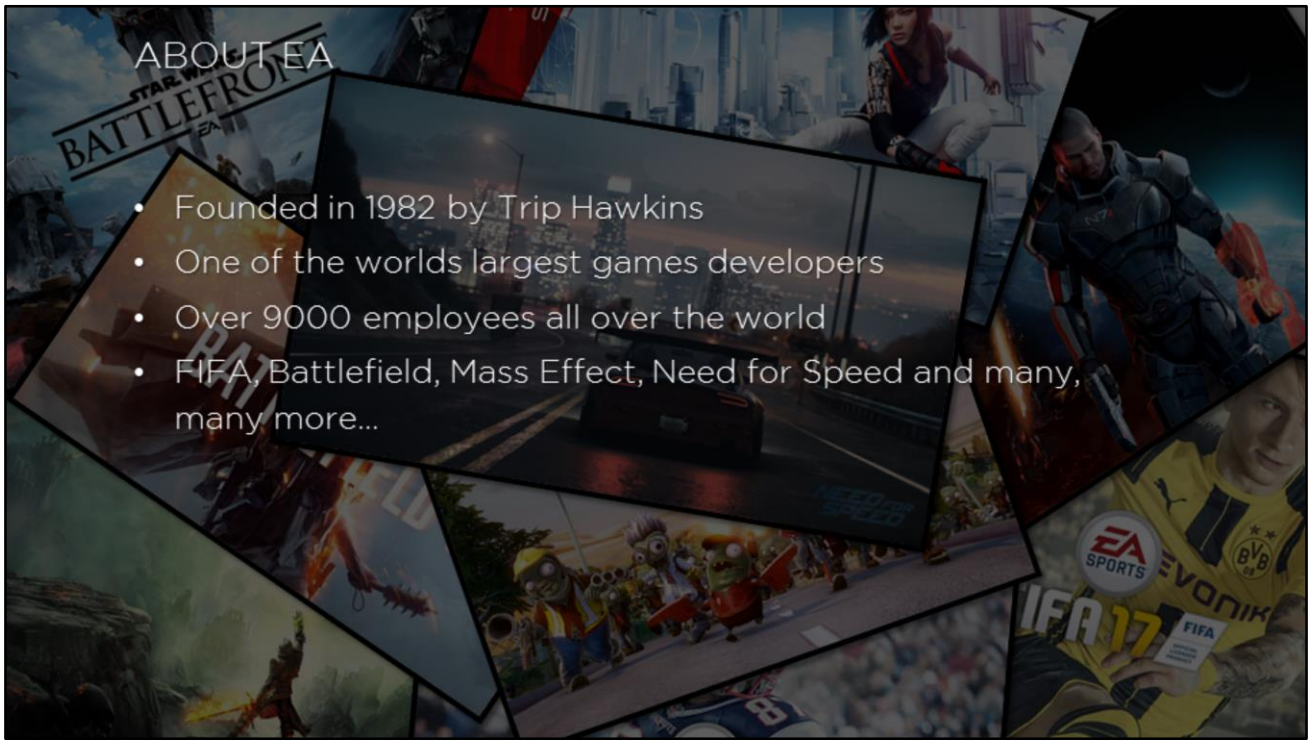
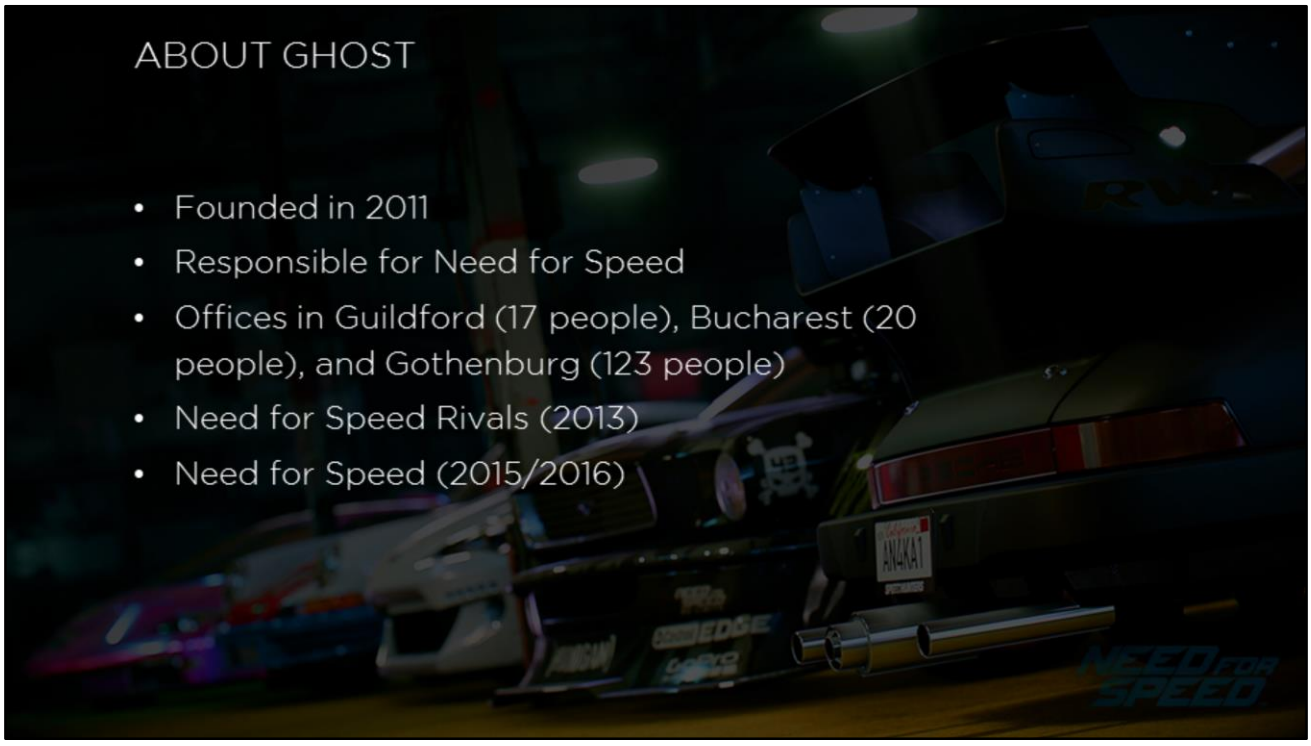

GHOST


GHOST
MATH IN GAMES
ANDREAS BRINCK



- In the mid eighties EA started making games for home computers
- Got into the console space in the nineties
- Through aquistions EA has turned into one of the world's largest publishers and developers
- Offices all over the world (Europe, North America, Oceania, South America and Asia)
- Very international working environment (on all levels)



- 9 people working from home initially, majority ex DICE Gothenburg
- Got entrusted with making the next NFS
- Rapid growth
- After ”exciting” process Rivals released in 2013 (360, PS3, XB1, PS4, and PC)

DEVELOPMENT

- Game code is C++
- Very large code base (probably millions of lines of code, but who's counting)
- Some tools are written in C# and python

WHAT MATH DO YOU NEED TO MAKE A GAME?

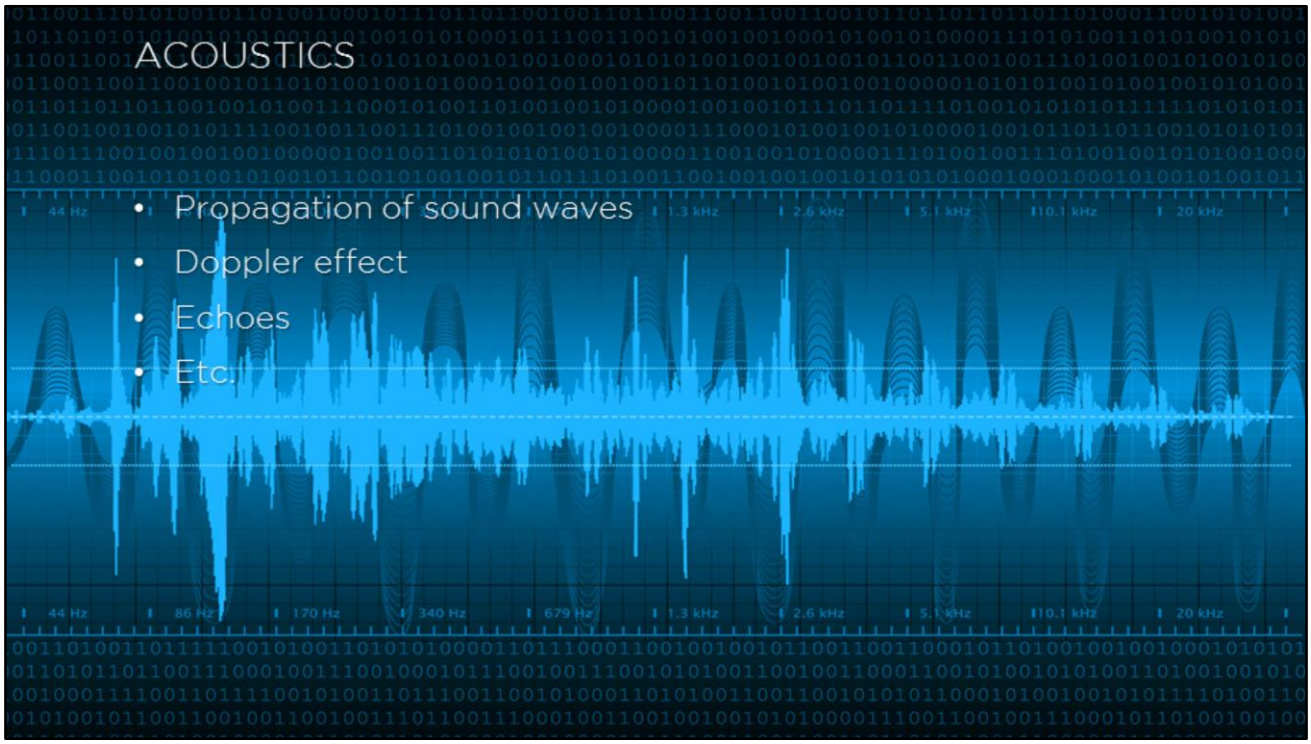
- Depends on the kind of game
- Our niche is action driving games
- Math is the result of simulating various physical phenomena
- We need a simulation of everything that is perceptible on a human level

<TRAILER OF NFS 2015>

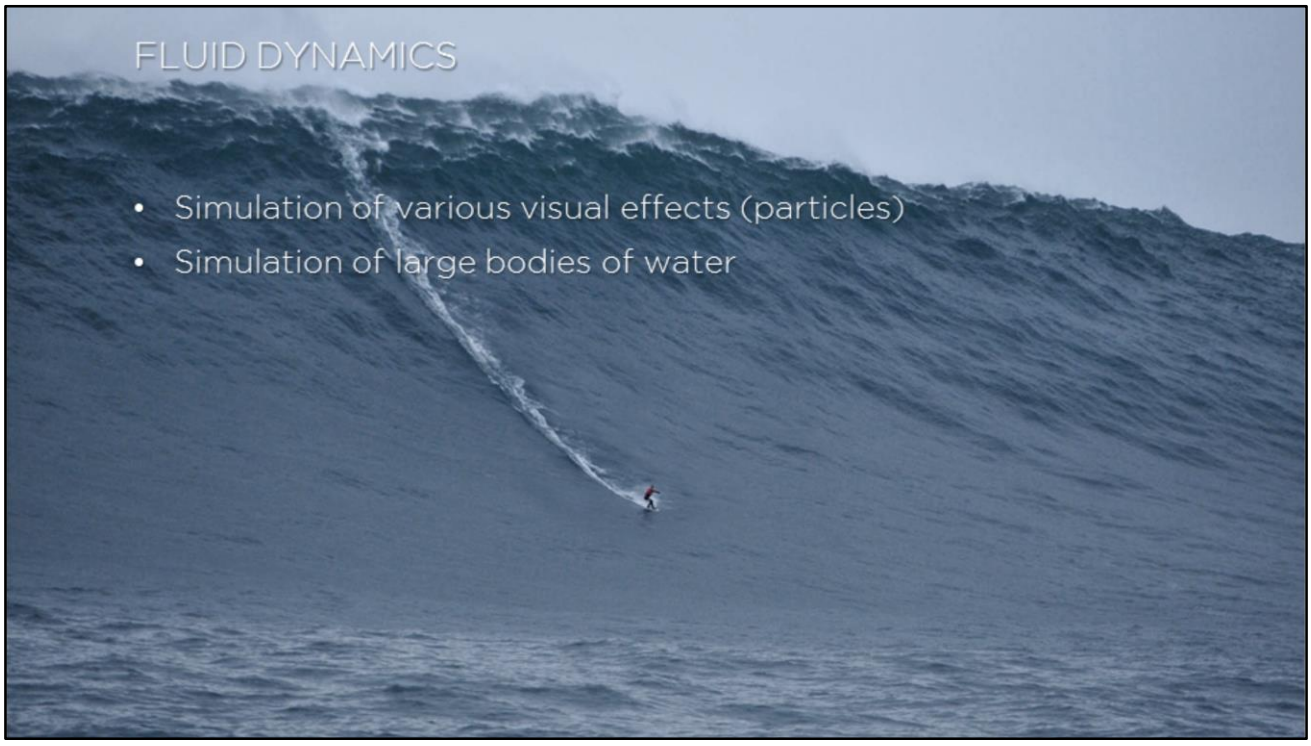

OPTICS
• How light interacts with surfaces and mediums in the game world
• Also simulation of some effects in the lens house of our virtual camera (and image sensor/eye)
GHOST
NEED FOR SPEED

MECHANICS

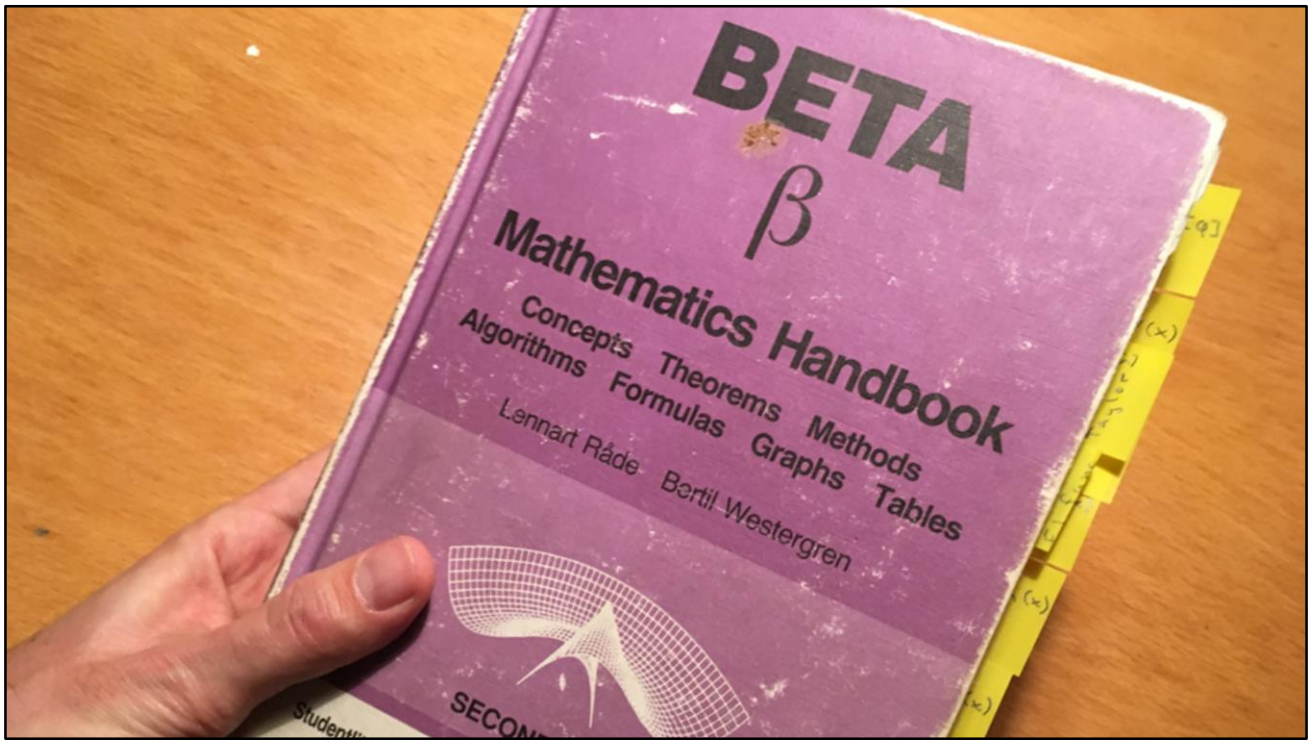
- Foundation of game world is a rigid body simulation
- Laws of Newton
- Constraints
- Various forces, i.e. collisions, friction, gravity



- We spend a lot of effort on getting the engine sounds correct in various environments



- Large bodies of water not used in NFS, but in other Frostbite games



What fields of mathematics does this lead to?

LINEAR ALGEBRA AND GEOMETRY

- By far the most used field of mathematics in games development

BASIC APPLICATION

- Coordinate systems and transforms between them
- Construction of geometry
- Equation systems
- Collision detection
- Visibility testing
- Reflection and refraction of light
- Many more...

SLIGHTLY MORE ADVANCED

- Principal Component Analysis (bounding boxes)
- Singular Value Decomposition (deformation)
- Large equation systems (and solvers)
- Fitting of data sets
- Convex hulls
- Delaunay triangulations
- Quaternions (interpolation of orientation)

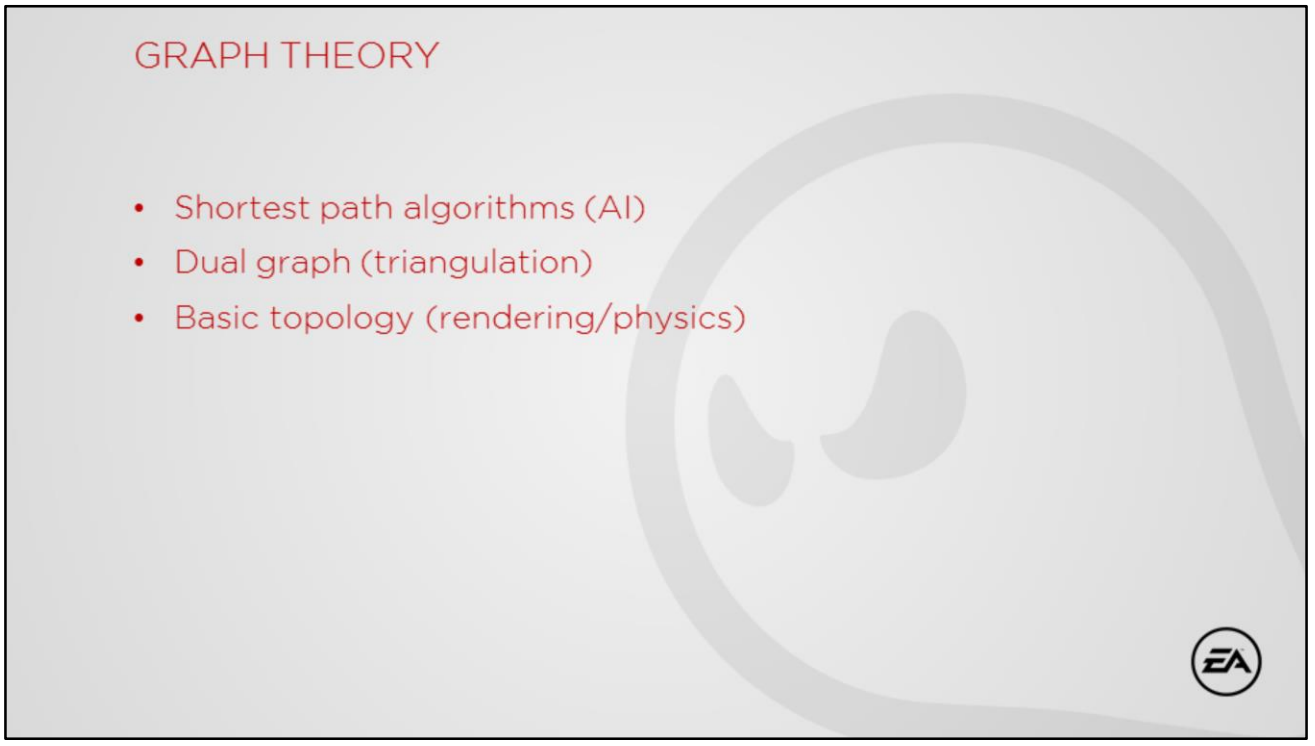
CALCULUS

- Curves and surfaces (rendering and AI)
- Integrals (numerical / analytic)
- Differentiation (rendering and AI)
- Series

$$L_0(\mathbf{x}, \omega_0, \lambda, t) = L_e(\mathbf{x}, \omega_0, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_0, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) d\omega_i$$

- Integrate over time/space
- Differentiation to find gradients, normals
- Series to approximate complex expressions
- Generally not that common

- Rigid body simulation and constraints
- Lemke's algorithm



- Delaunay triangulation
- Some rendering and physics algorithms have special demands on topology

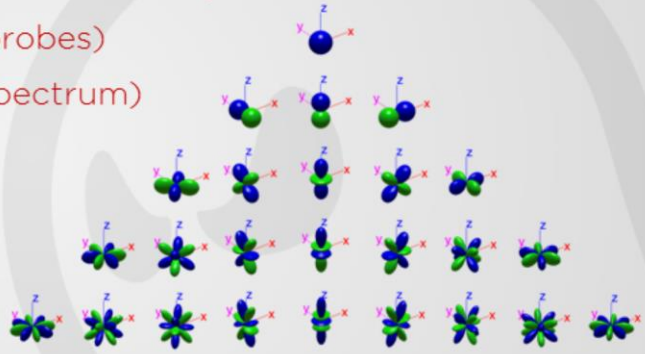
STATISTICS AND PROBABILITY

- Importance sampling (rendering)
- Distributions (rendering/gameplay/AI)
- Sequences (ditto)

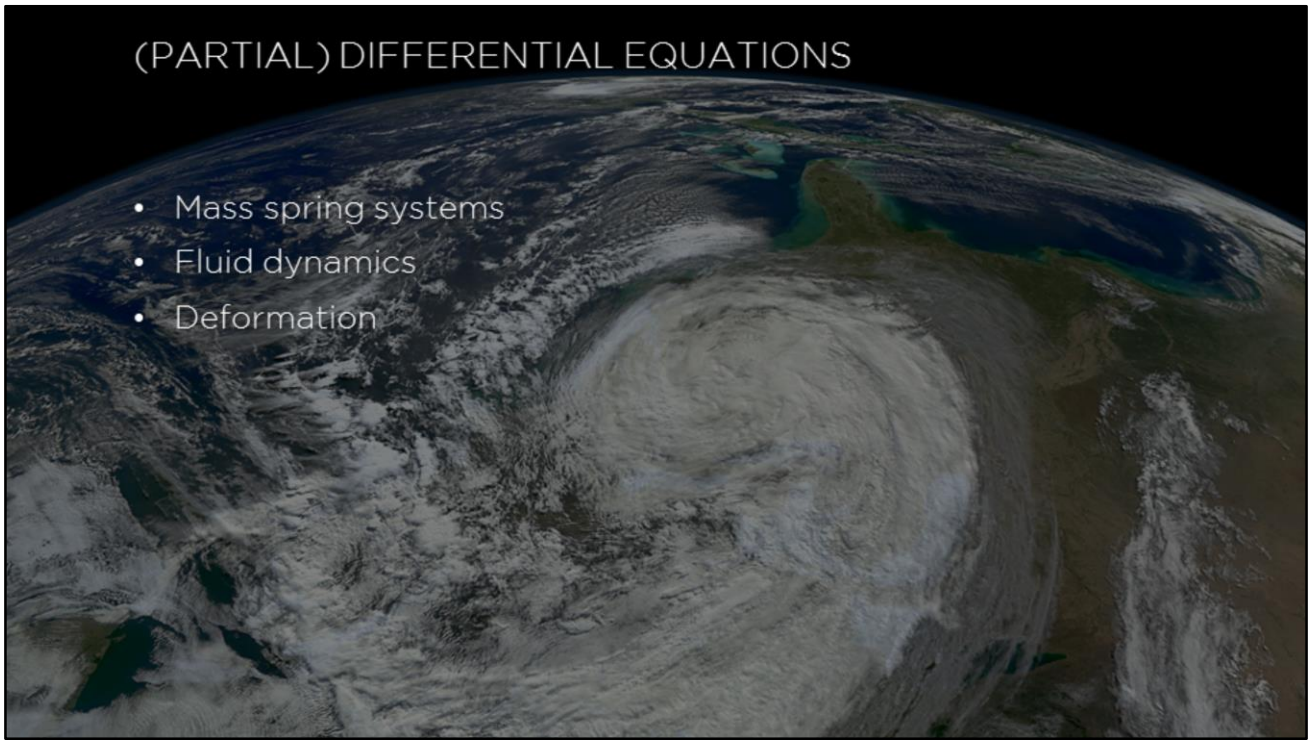
- Importance sampling used in numerical integration of some rendering effects
- Sequences and distributions used in rendering algorithms (TAA, placement of objects, sampling strategies)
- Also used in gameplay systems

FOURIER ANALYSIS

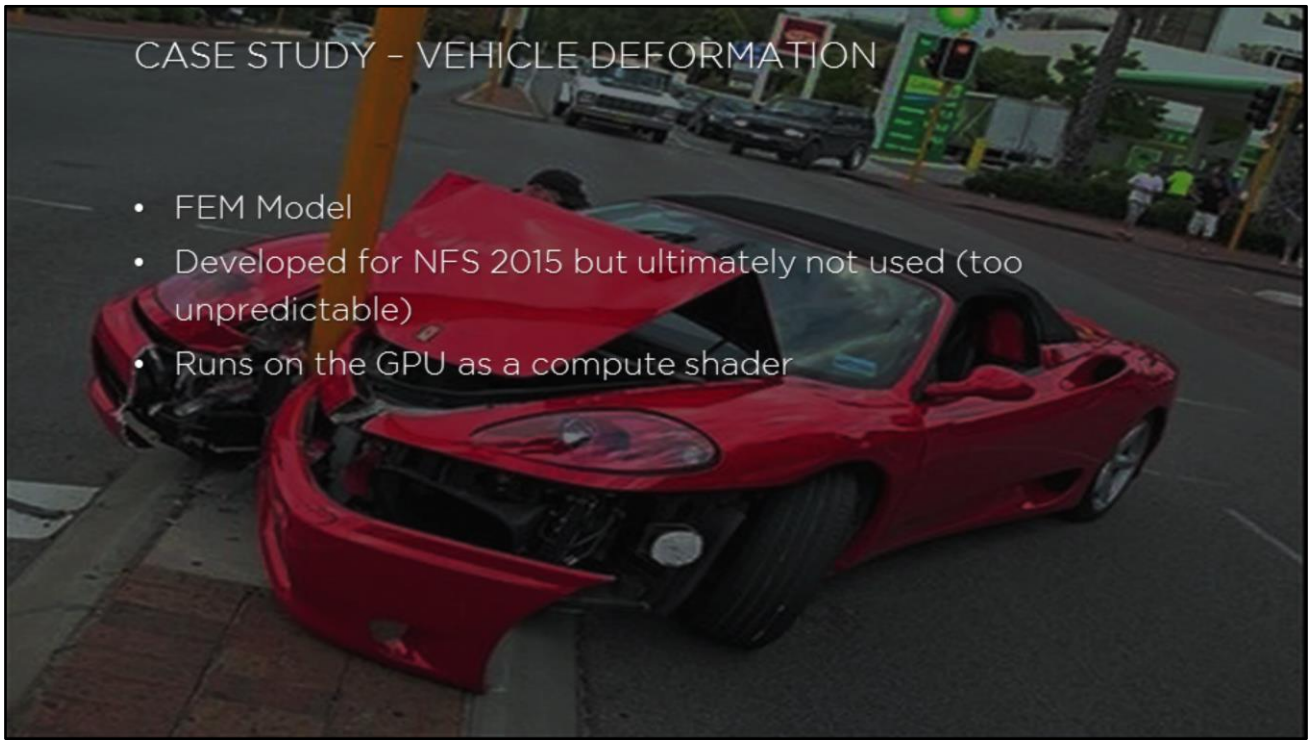
- Compression of data (DCT, i.e. JPEG etc.)
- Spherical harmonics (light probes)
- Ocean simulation (Phillips spectrum)
- Water interaction
- Filters (rendering/audio)



- Spherical harmonics for light probes is also a form of compression over a spherical domain


(PARTIAL) DIFFERENTIAL EQUATIONS
• Mass spring systems
• Fluid dynamics
• Deformation

- Important to get right
- Floating point numbers are not magic
- Beware of cancellation
- Epsilon not magic bullet either

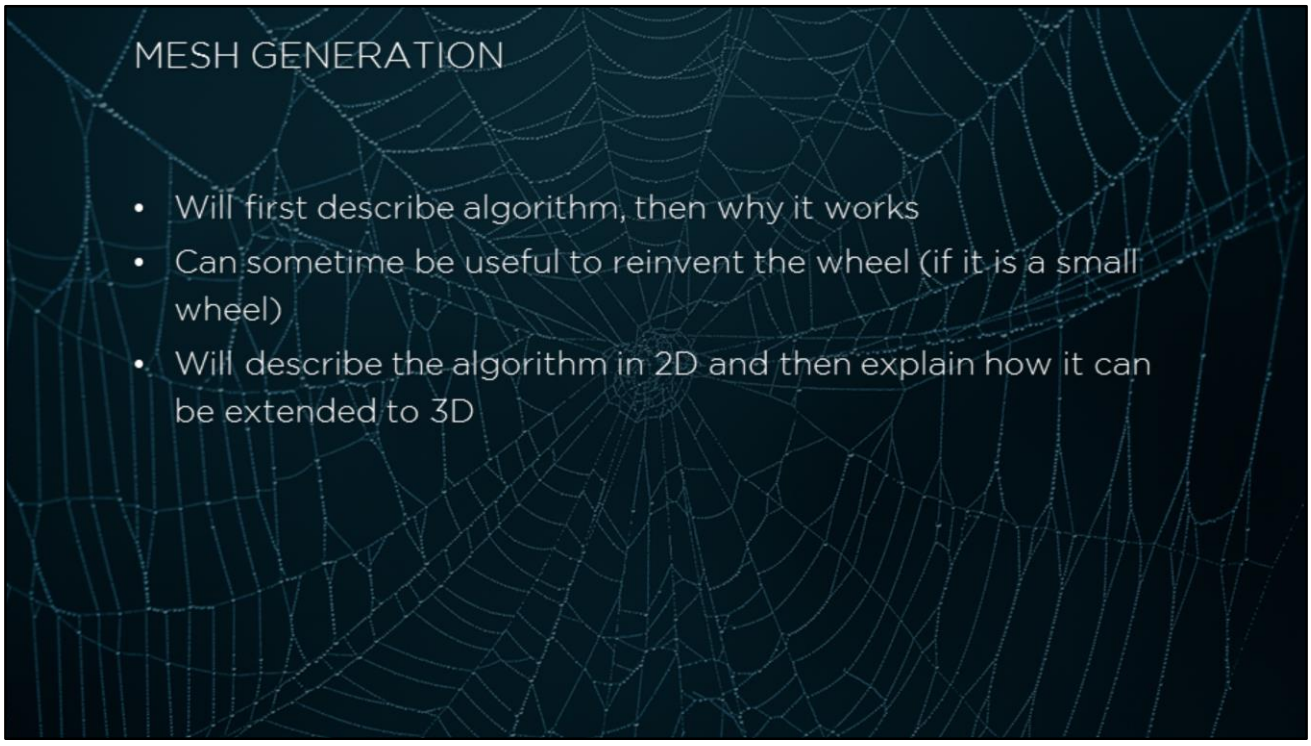


- Artists like predictable unpredictable ☺
- Real real-time, not academic real-time (budget ~2ms of GPU time)
- See “Real-Time Deformation and Fracture in a Game Environment” by Parker and O’Brien for more details

IDEA

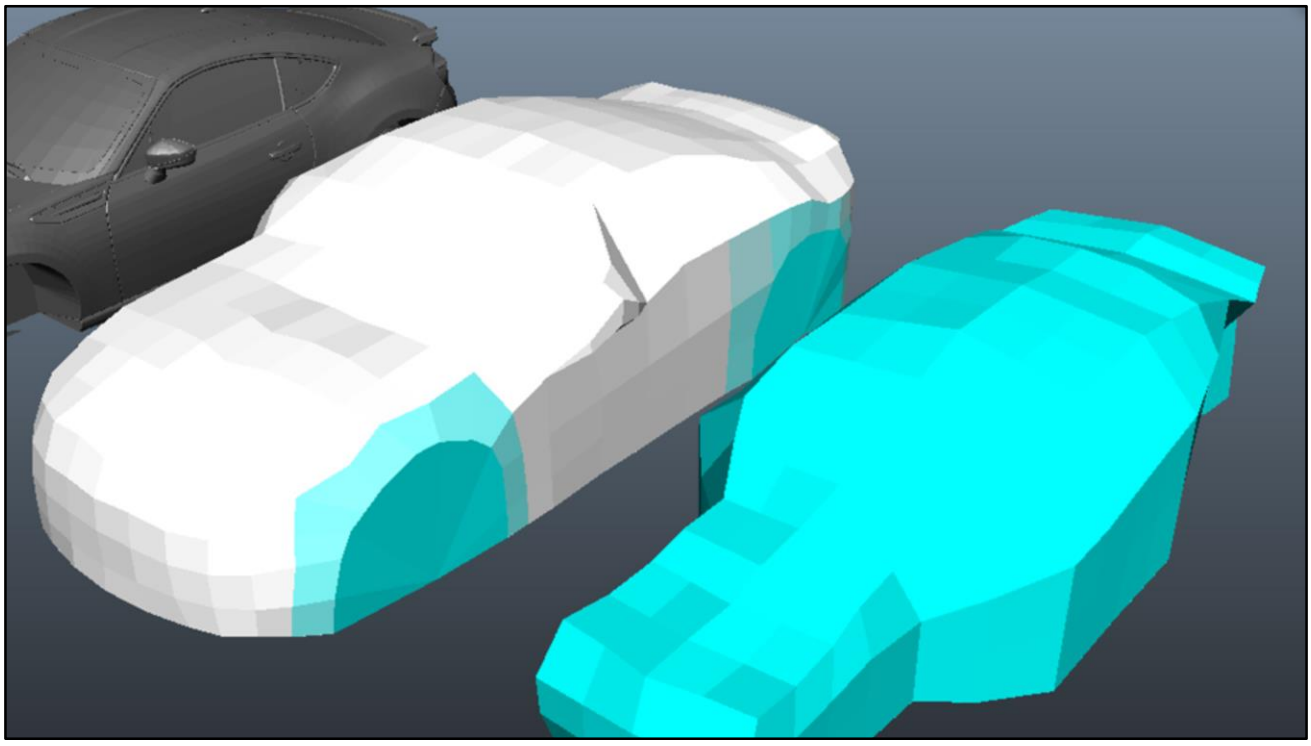
- Create a low fidelity control mesh
- Solve the equation system resulting from the discretized model (FEM)
- Use the control mesh to deform the high fidelity mesh

<DEMONSTRATION MOVIE OF DEFORMATION SYSTEM>

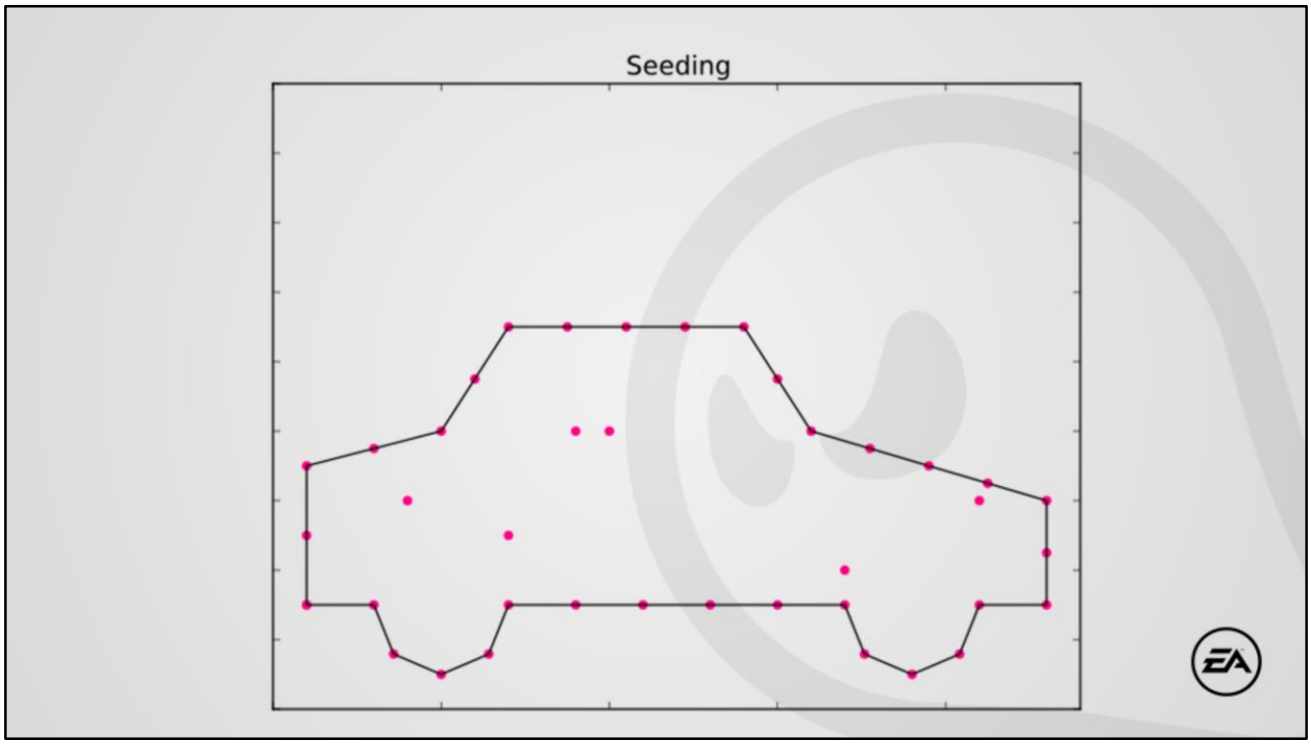

MESH GENERATION
• Will first describe algorithm, then why it works
• Can sometime be useful to reinvent the wheel (if it is a small wheel)
• Will describe the algorithm in 2D and then explain how it can be extended to 3D

INPUT DATA

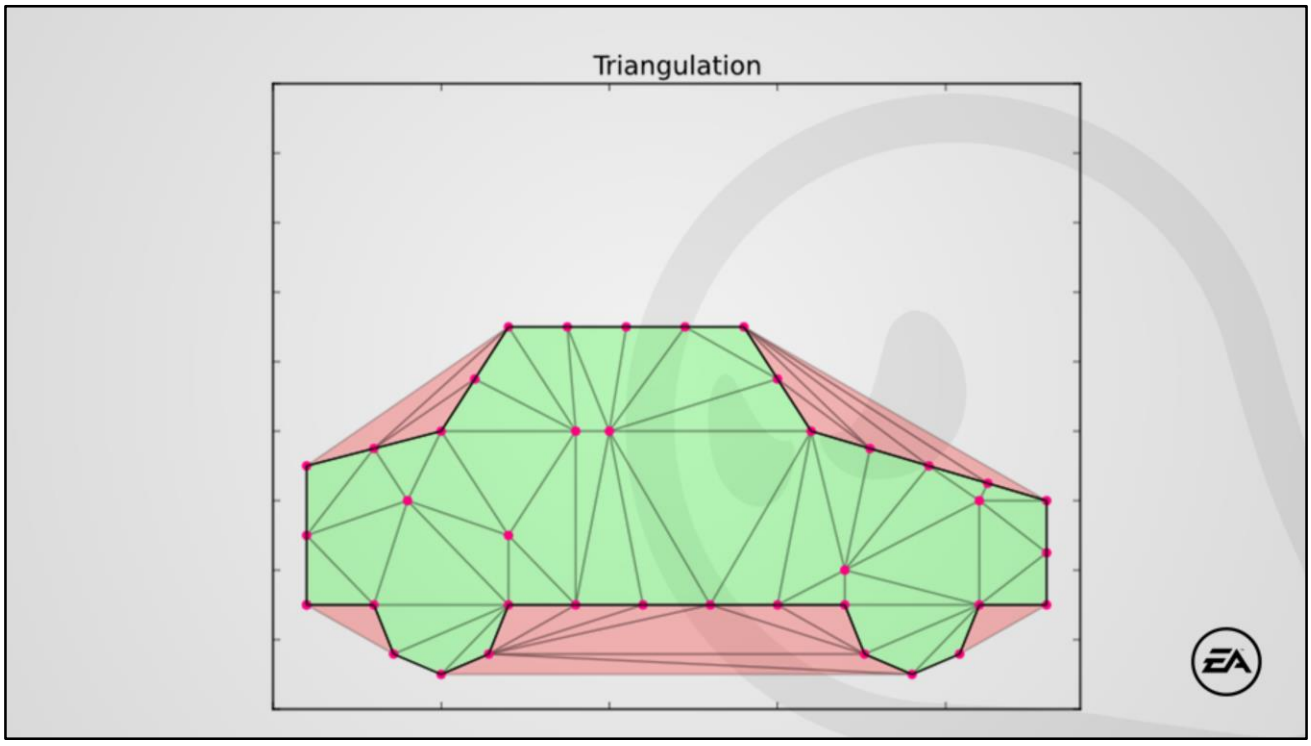
- The input data is a polygonal control shell provided by our artists
- Attributes can be specified with vertex colors



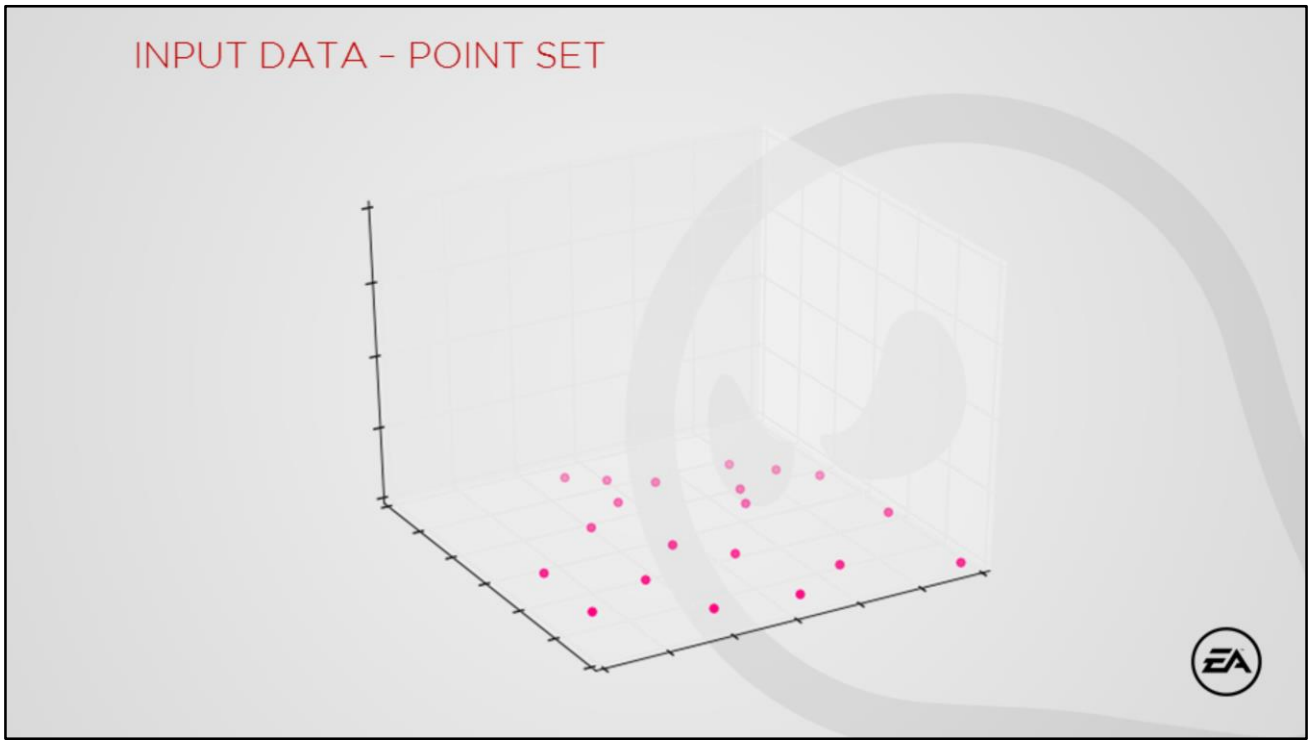
- Nodes can be locked in place
- Control mesh can contain several shells
- We want to keep the engine block, wheel houses, and roll cage intact (shell shown on the right)
- Setting the red component of vertex to zero locks it in place



- First step is to insert a number of random points inside the control mesh
- The number of points determine the size of the system that we'll eventually have to solve

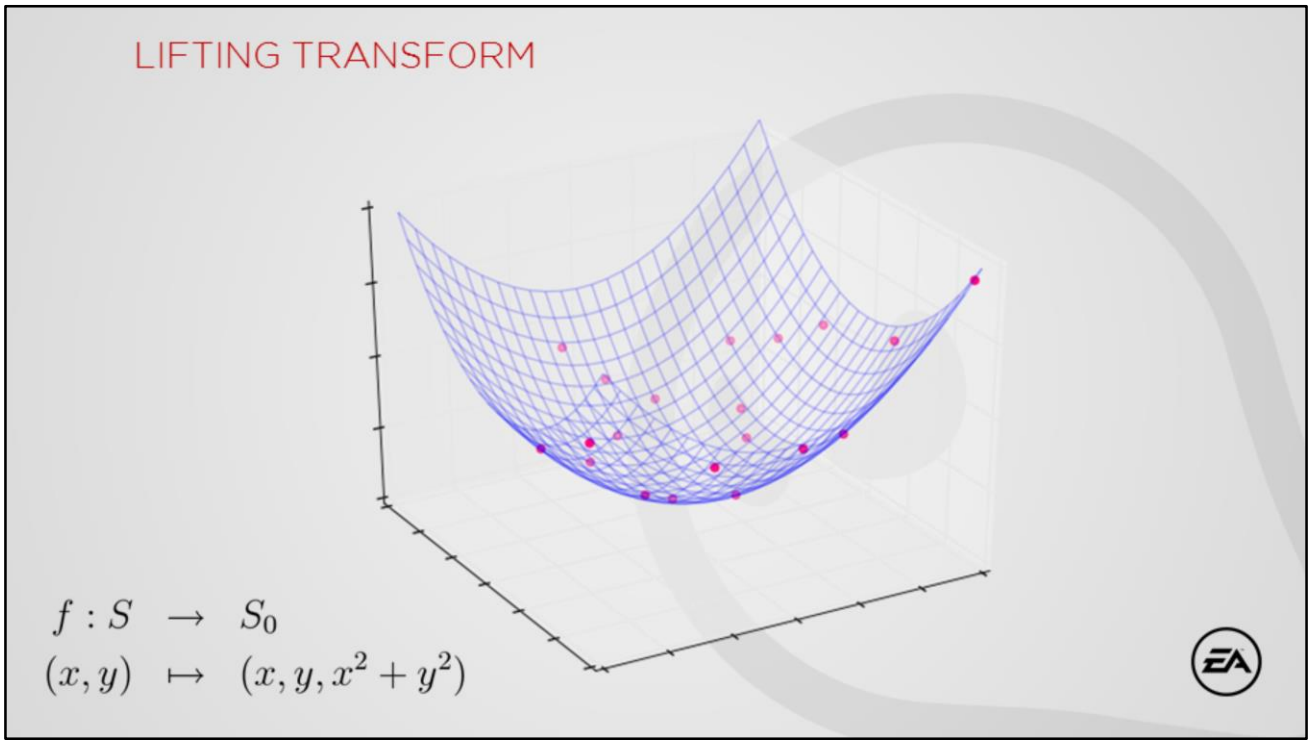


- Next step is to generate a Delaunay triangulation
- A Delaunay triangulation is a triangulation where the circle that circumscribe each triangle doesn't contain any other vertex of the input data set



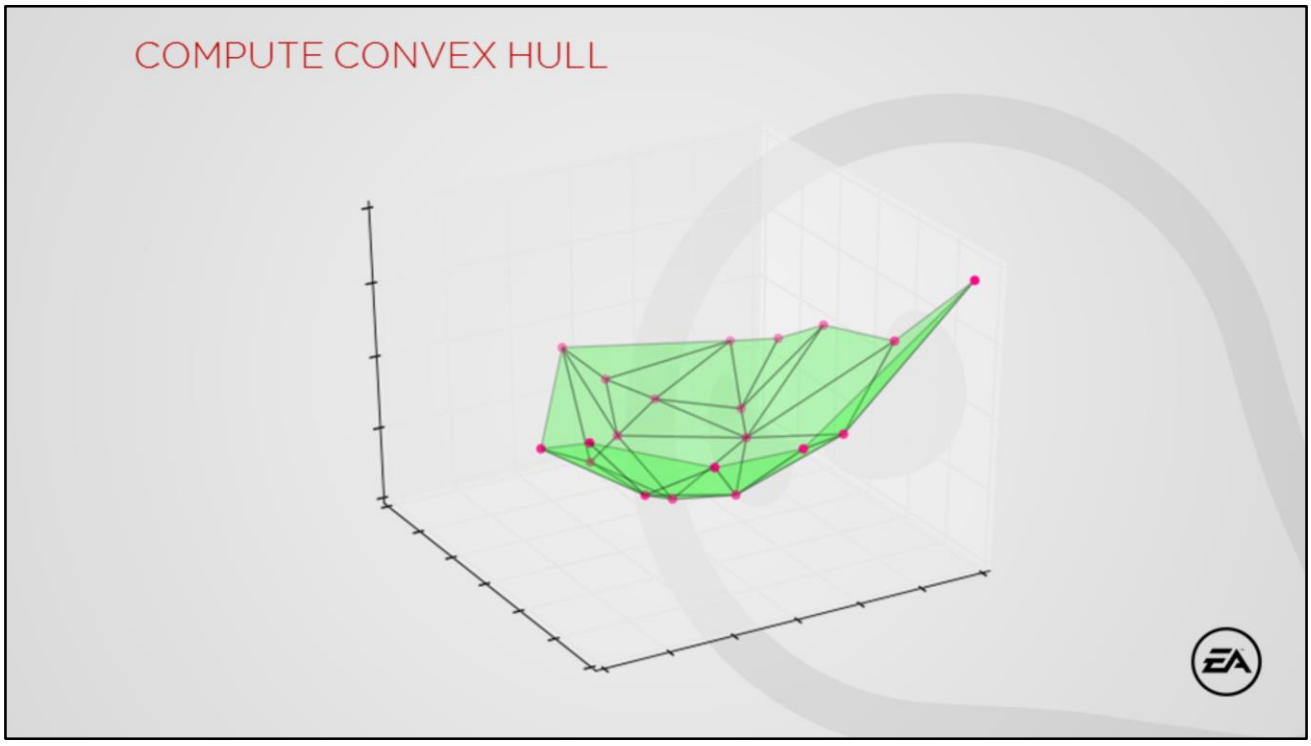
- So how do we generate the Delaunay triangulation?

LIFTING TRANSFORM



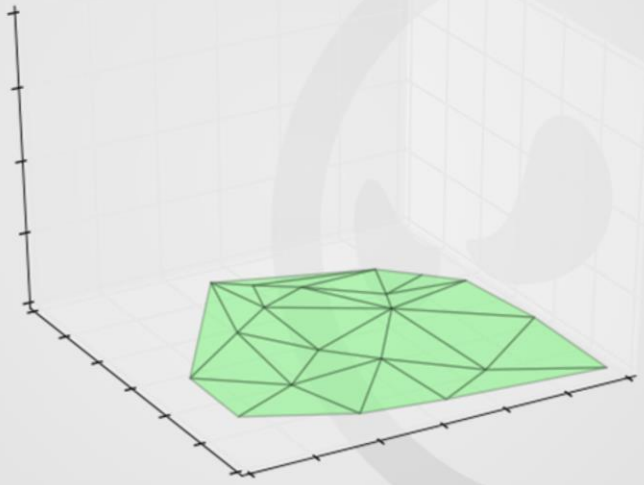
- Apply a lifting transform, moves data from dimension N to dimension N + 1 (2D to 3D in this case)

COMPUTE CONVEX HULL



- We use the quickhull algorithm to find the hull
- Points represented as rational numbers (could have used CGAL or predicates as well) for exact tests
- Straight up floating point numbers are kryptonite to computational geometry

PROJECT LOWER PART



- When we project back we get the triangulation

MOTIVATION WHY THIS WORKS

- Claim: given *p, q, r*, and *s*, *s* lies inside the circumcircle of *p, q,* and *r* iff *s0* is below the plane formed by *p0, q0,* and *r0*.
- Find plane equation for plane passing and tangent to lifted *(a, b)*:

$$
\begin{aligned}
z &= 2ax + 2by + k \rightarrow \\
a^2 + b^2 &= 2a^2 + 2b^2 + k \rightarrow \\
k &= -(a^2 + b^2) \rightarrow \\
z &= 2ax + 2by - (a^2 + b^2)
\end{aligned}
$$

- Idea, establish connection between no point inside circle in 2D, and no point below plane in 3D

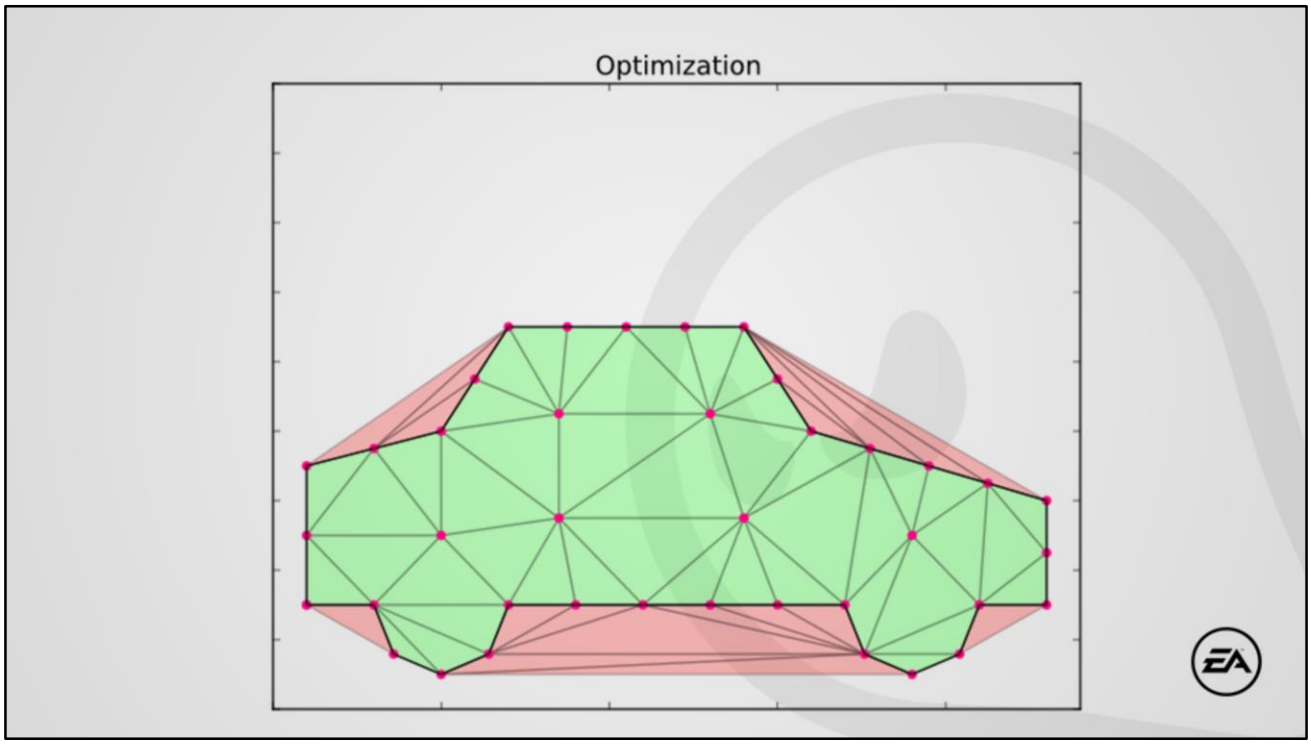
MOTIVATION WHY THIS WORKS

- Offset the plane vertically:

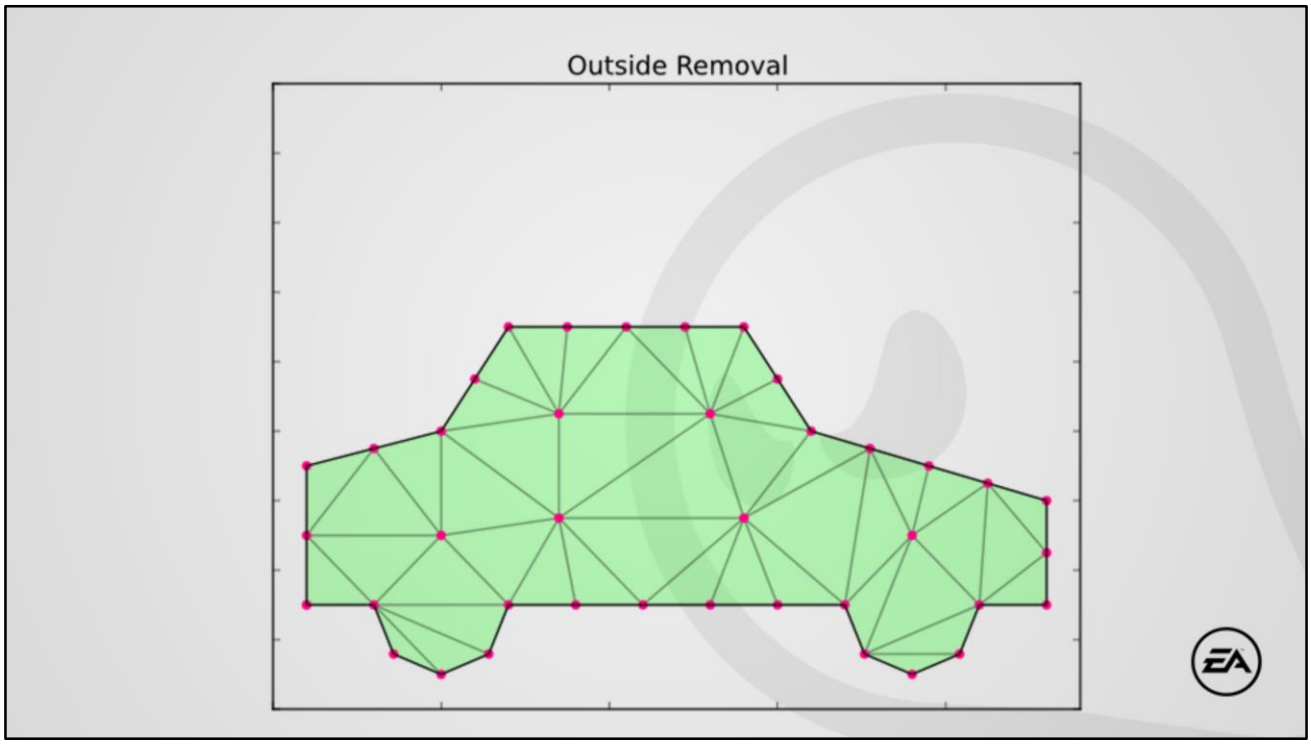
$$
\begin{aligned}
z &= 2ax + 2by - (a^2 + b^2) + \rho^2 \rightarrow \\
x^2 + y^2 &= 2ax + 2by - (a^2 + b^2) + \rho^2 \rightarrow \\
(x - a)^2 + (y - b)^2 &= \rho^2
\end{aligned}
$$

- Conclusion projection of intersection is a circle centered around *(a, b)* with radius *rho*
- Points on the parabola below the plane is inside the circle

- The projection of the intersection of the plane passing through one of the faces of the convex hull and the parabola defines a circle in the 2D plane. The only projections of points on the parabola that end up inside this circle are the ones that are below the plane, but since the plane is part of the convex hull, there are no such points, which is what we wanted to show



- If we're not happy with the shape of the elements we move the nodes and triangulate again
- Wash, rinse, repeat, until satisfied (or for maximum number of iterations)



- Finally we remove the elements outside of the shells defined by the artists

HOW TO EXTEND TO 3D

- Lift vertices from 3D to 4D, w = x2 + y2 +z2
- In the 2D case the faces of the convex hull are 3D triangles, in the 3D case the faces of the convex hull are 4D tetrahedrons
- Exact same proof can be made for why this works in 3D, but in this case the projection of the intersection and the plane will be a sphere (In 3D the Delaunay condition is that the circumsphere formed by the 4 tetrahedron vertices doesn't contain any other vertex)

- You need good geometric intuition to work with this

COUPLED SYSTEM OF EQUATIONS

$$M\mathbf{a} + C\mathbf{v} + K(\mathbf{x} - \mathbf{u}) = \mathbf{f}_{\mathbf{ext}}$$

- a, v, x, and f are vectors of vectors
- M, C, and K are matrices of tensors
- M is the mass matrix, mass is attached to nodes (mass of each tetrahedron split evenly among nodes)

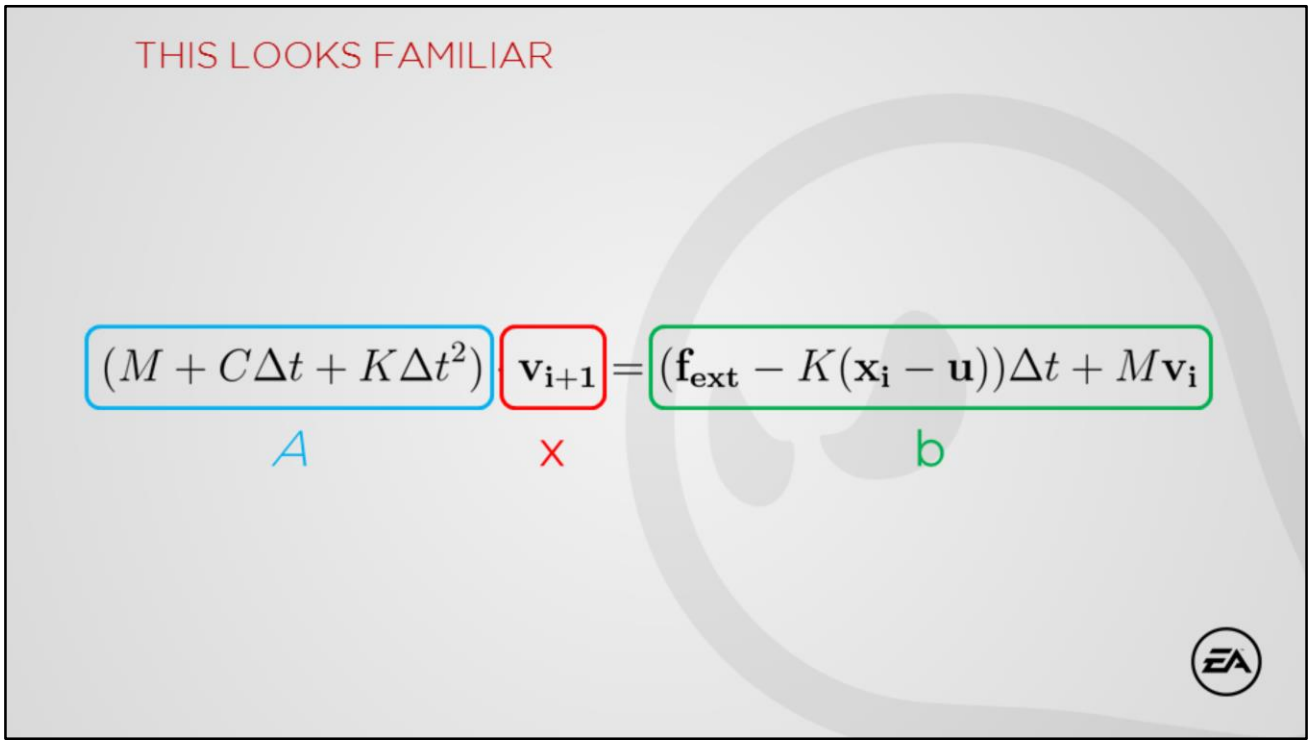
INTEGRATOR

- Semi-implicit Euler method

$$\begin{aligned}\mathbf{v}_{i+1} &= \mathbf{v}_i + \mathbf{a}_{i+1}\Delta t \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{v}_{i+1}\Delta t\end{aligned}$$

$$\begin{aligned}M\mathbf{a}_{i+1} + C\mathbf{v}_{i+1} + K(\mathbf{x}_{i+1} - \mathbf{u}) &= \mathbf{f}_{ext} \rightarrow \\ M\mathbf{a}_{i+1}\Delta t + C\mathbf{v}_{i+1}\Delta t + K(\mathbf{x}_{i+1} - \mathbf{u})\Delta t &= \mathbf{f}_{ext}\Delta t \rightarrow \\ M(\mathbf{v}_{i+1} - \mathbf{v}_i) + C\mathbf{v}_{i+1}\Delta t + K(\mathbf{x}_i + \mathbf{v}_{i+1}\Delta t - \mathbf{u})\Delta t &= \mathbf{f}_{ext}\Delta t \rightarrow \\ (M + C\Delta t + K\Delta t^2)\mathbf{v}_{i+1} &= (\mathbf{f}_{ext} - K(\mathbf{x}_i - \mathbf{u}))\Delta t + M\mathbf{v}_i\end{aligned}$$

- We want to solve for v_{i + 1}



- Equation system

HOW TO COMPUTE K

- Starting point:

$$\sigma = \lambda tr(\varepsilon)I + 2\mu\varepsilon$$

- Hooke's law in 3D
- my and lambda can be derived from E (young modulus) and Poisson's ratio (see Lamé parameters)
- C is a damping constant and can be calculated from K and M

- Hooke's law relates strain to tension.
- Generalized spring

HOW DO WE FIND THE STRAIN?

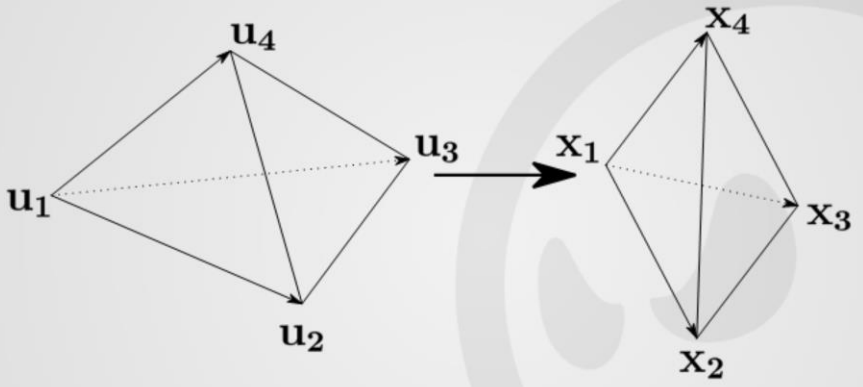
- Cauchy's infinitesimal strain tensor

$$\varepsilon = \frac{1}{2}(F + F^T) - I$$

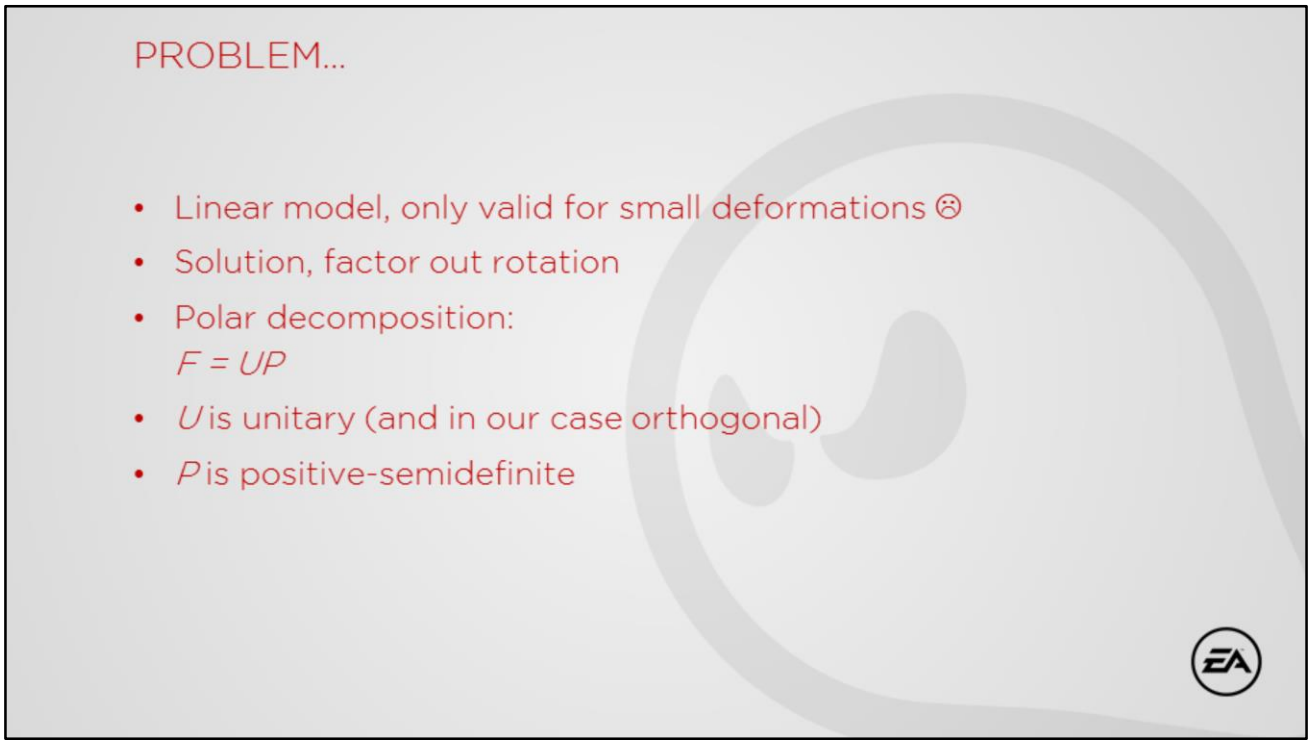
- F is the deformation gradient

- Valid for small displacements
- Not rotationally invariant

WHAT IS THE DEFORMATION GRADIENT



$$F = [\mathbf{x_2} - \mathbf{x_1}, \mathbf{x_3} - \mathbf{x_1}, \mathbf{x_4} - \mathbf{x_1}][\mathbf{u_2} - \mathbf{u_1}, \mathbf{u_3} - \mathbf{u_1}, \mathbf{u_4} - \mathbf{u_1}]^{-1}$$



- This means we have to rebuild stiffness matrix in each iteration though

POLAR DECOMPOSITION

- Singular value decomposition:

$$F = UP \rightarrow \{SVD\}$$

$$F = W\Sigma V^T \rightarrow$$

$$U = WV^T$$

- Handles inverted elements as well
- Incremental algorithm of McAdams

REMOVE ROTATION FROM STRESS CALCULATION

- Corotated deformation gradient:

$$\tilde{F} = U^{-1}F = U^{-1}UP = P$$

- Corotated deformation strain:

$$\tilde{\varepsilon} = \frac{1}{2}(\tilde{F} + \tilde{F}^T) - I$$

- Corotated stress:

$$\tilde{\sigma} = \lambda tr(\tilde{\varepsilon})I + 2\mu\tilde{\varepsilon}$$

- Strain in space of rotated element
- Note for further slides, for orthonormal transform the inverse is the same as the transpose

JACOBIAN OF STRESS FORCE

- Force exerted on node i (Caveat Emptor!):

$$\mathbf{f_i} = U\tilde{\sigma}\mathbf{n_i}$$

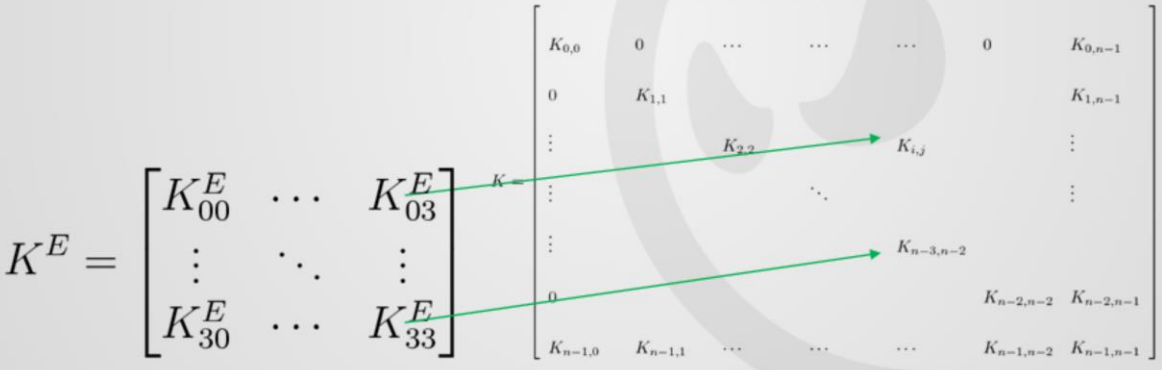
- K^E is the Jacobian of f with respect to the node displacements:

$$K_{ij}^E = U(\lambda \mathbf{n_i}\mathbf{n_j^T} + \mu(\mathbf{n_i}, \mathbf{n_j}) + \mu \mathbf{n_j}\mathbf{n_i^T})U^T$$

- Intuitively transform displacement into rotated system, compute force, and rotate back
- Not sure about the force equation, it is not exactly consistent with what we use for our virtual forces to enforce plasticity, but can't remember why and didn't have time to reinvestigate for this presentation ¯_(ツ)_/¯

ASSEMBLE THE GLOBAL STIFFNESS MATRIX

- Each K^E will contribute to 16 elements in the global stiffness matrix
- Each element in the global matrix corresponds to an edge and will get contributions from several mesh elements



- K is sparse and the structure never changes
- We use a block compressed row format (spans) to store elements
- Each element in global K is a sum of tensors from the contributing edges
- System is originally symmetric and positive definite

CONSTRAINTS

- We can fixate nodes by forcing velocity to 0
- Destroys symmetry of A but not a big problem

$$\begin{bmatrix} A_{0,0} & \cdots & A_{0,n-1} \\ \vdots & \ddots & \vdots \\ 0 & 0 \; 0 \; I & 0 \\ \vdots & \ddots & \vdots \\ A_{n-1,0} & \cdots & A_{n-1,n-1} \end{bmatrix} \mathbf{v_{i+1}} = \begin{bmatrix} b_0 \\ \vdots \\ 0 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

- Position constraint can be turned into a velocity constraint
- Unfortunately this destroys the symmetry of the matrix
- No way to directly implement contact constraints

SOLVER

- Conjugate gradient
- Jacobi preconditioner

$$P^{-1}(A\mathbf{v_{i+1}} - \mathbf{b}) = 0$$

$$P = M = \begin{bmatrix} M_0 & 0 & \cdots & 0 \\ 0 & M_1 & & \\ \vdots & & \ddots & \\ 0 & & & M_{n-1} \end{bmatrix}$$

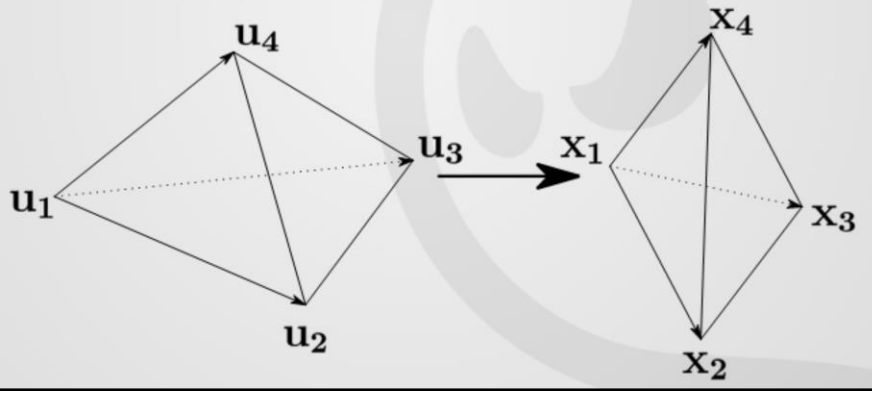
- Conjugate gradient theoretically requires the matrix to be symmetric, but in practice it works well with non symmetric matrices as well
- The term stiff as used with numerical problems comes from the fact that solving systems with stiff materials converges slower than soft materials
- We use a cheap diagonal Jacobi preconditioner to improve the condition of the system

WHAT ABOUT PLASTICITY?

- Will not cover in detail
- Basic idea is to separate the total strain into one plastic and one elastic component
- Elastic part is what's used in the force calculation
- Plastic part updated when material yield threshold is exceeded
- Modeled as virtual internal forces to avoid updating data structures

DEFORMATION OF VISUAL MESH

- Remember this?
- Each vertex in visual mesh holds an index to the element that contains it

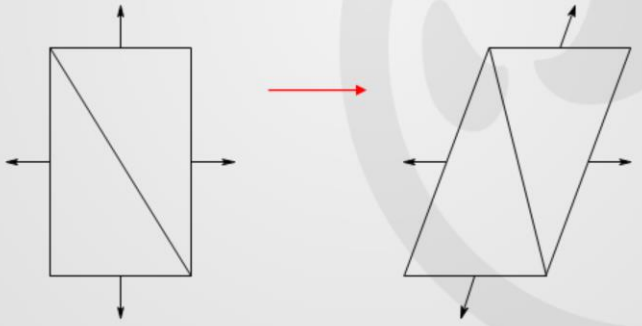


DEFORMATION OF POSITIONS

$$\tilde{\mathbf{v}} = \begin{bmatrix} \mathbf{x_2} - \mathbf{x_1} & \mathbf{x_3} - \mathbf{x_1} & \mathbf{x_4} - \mathbf{x_1} & \mathbf{x_1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u_2} - \mathbf{u_1} & \mathbf{u_3} - \mathbf{u_1} & \mathbf{u_4} - \mathbf{u_1} & \mathbf{u_1} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{v}$$

DEFORMATION OF NORMALS (AND TANGENTS ETC.)

- Initial attempt was to transform with same matrix as for positions (normals are homogenous coordinates with w = 0)
- Didn't look good though:



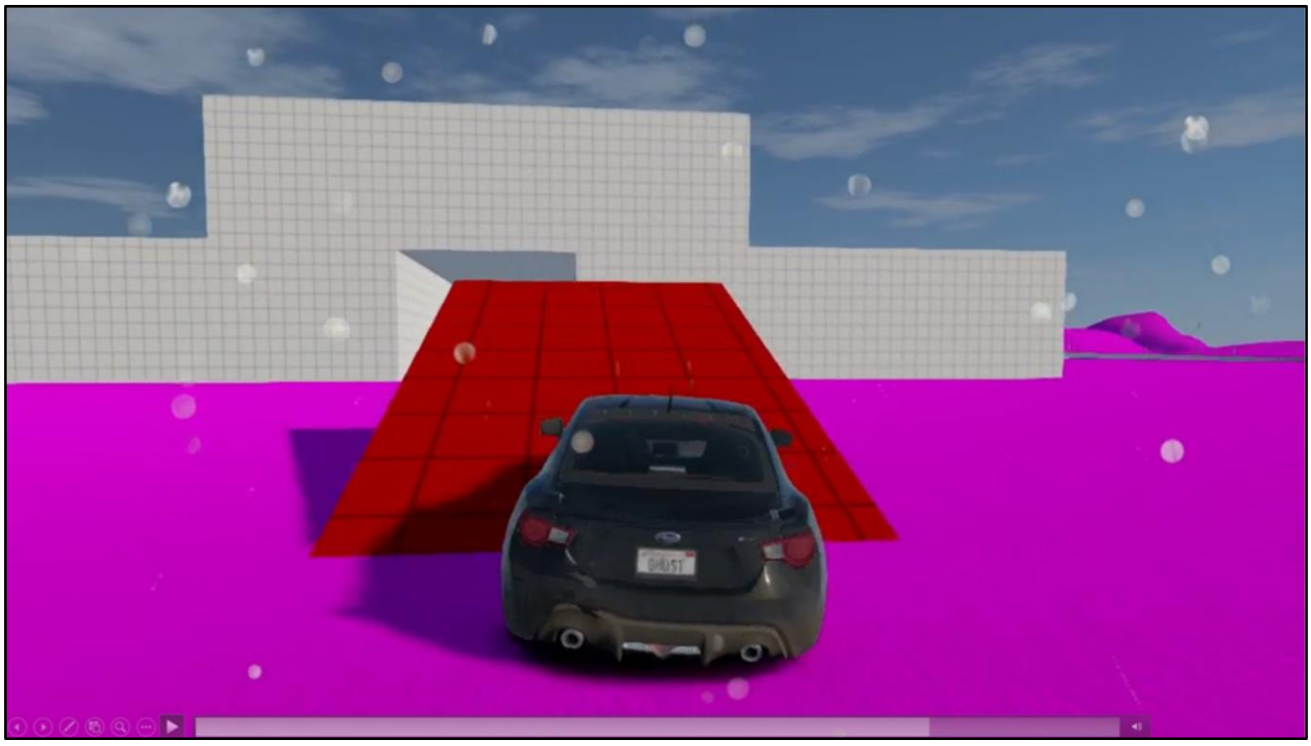
- The normals become skewed in relation the surface
- Same thing with tangents

DEFORMATION OF NORMALS

- Solution: use the rotation extracted in the polar decomposition of the deformation gradient

$$\tilde{\mathbf{n}} = U\mathbf{n}$$

- We already have this data available
- In addition to the whole vehicle deformation, we also have a system of “deformation sensors” that can be used to do a rigid transformation of things like head lights etc. Can also be used to trigger particle effects



<MOVIE DEMONSTRATING THE SYSTEM IN GAME>

HOW TO GET A JOB IN GAMES

- Learn C++
- Learn math and physics
- Practice!

abrinck@ghostgames.com
@andreas_brinck

THANKS TO

- Andreas Lindqvist, awesome programmer at Ghost who implemented the solver in a compute shader
- The vehicle artists at Ghost, in particular Christopher Kristiansson
- Anders Logg, professor of computational mathematics at Chalmers University of Technology