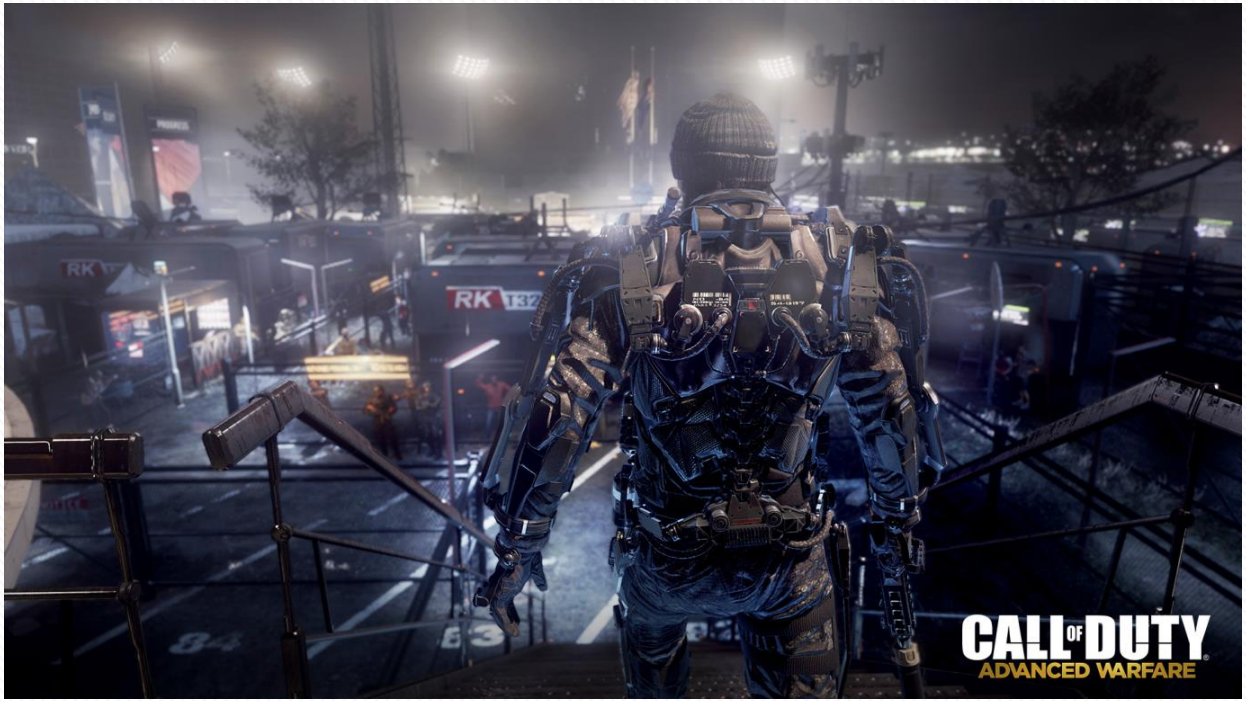
Approximate models for physically based rendering

Michał Iwanicki
Angelo Pesce

Activision

Introduction

Physically-based models: more and more popular



Introduction

- Complex!
  - No closed form solutions
  - Real time applications can often afford only the simplest cases

Introduction

- How can we use them in real time?
- Model and approximate!
  - Looks just as good (*)
  - Way cheaper at runtime

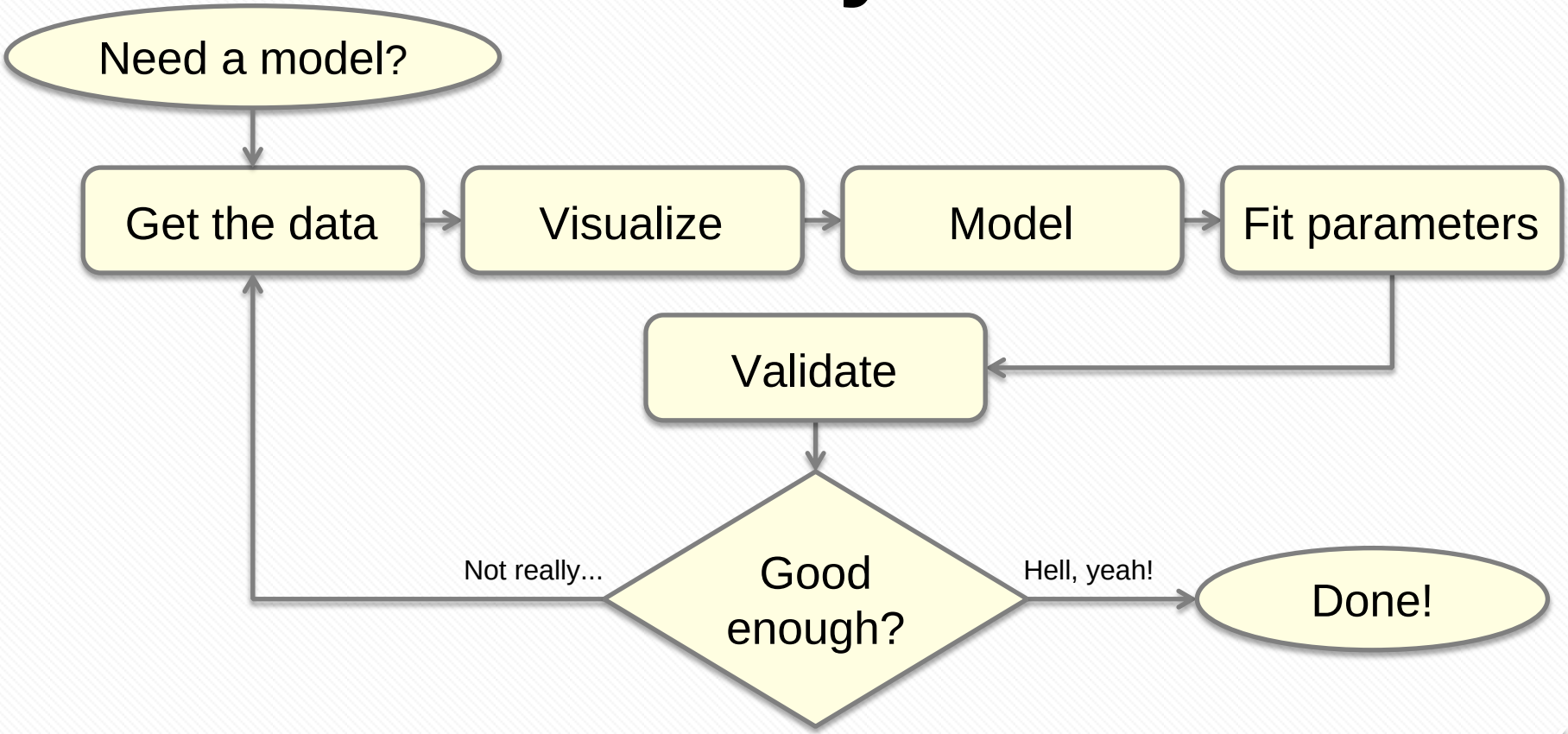
(*) well, ok, almost ;-)

Introduction

- Creating approximate models
  - Trial-and-error process
  - Practice makes perfect
- Sharing our experience
  - Basic guidelines (*)
  - Case studies!

(*) No worries, there's more theory in the course notes

The cycle

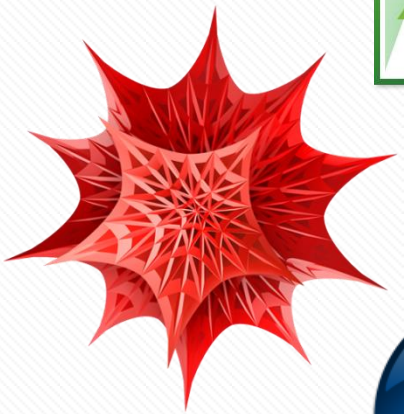

Need a model?
Get the data
Visualize
Model
Fit parameters
Validate
Good enough?
Not really...
Hell, yeah!
Done!

1. Acquire the data

- From an existing model:
  - E.g. compute the value of the BRDF for every possible combination of input parameters
  - E.g. compute the desired values using costly precomputations
- From reality:
  - Scan/measure

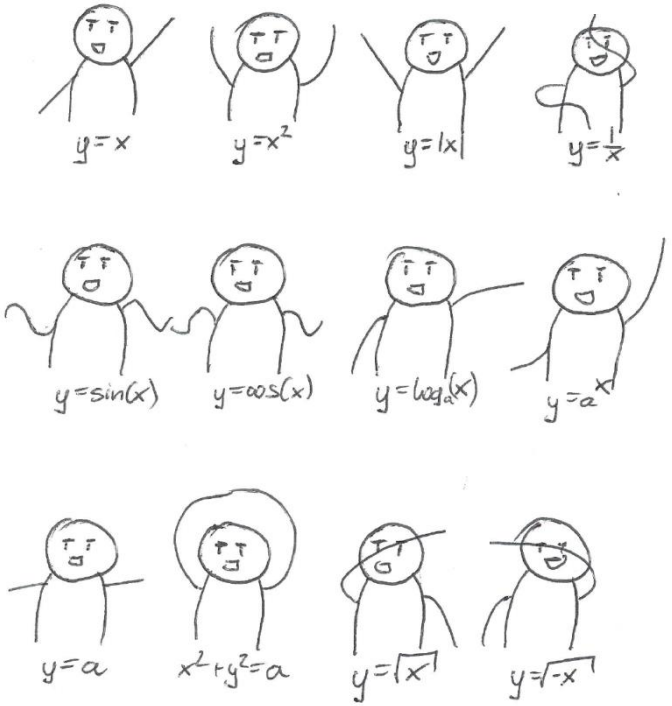
2. Visualize

- Know your enemy, visualize as much as possible
  - Excel, Mathematica, SciPy
  - Render cross sections through different dimensions
  - Make visualizations interactive when possible



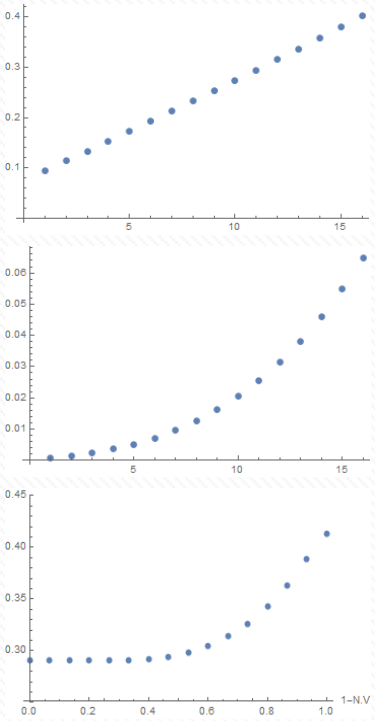
3. Create the approximate model

- The tricky part :)
- Start from visualized data
- Cheat sheet



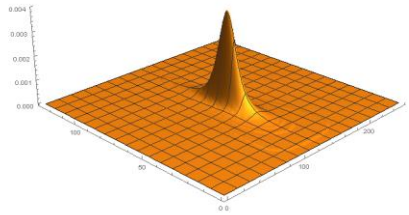
3. Create the approximate model

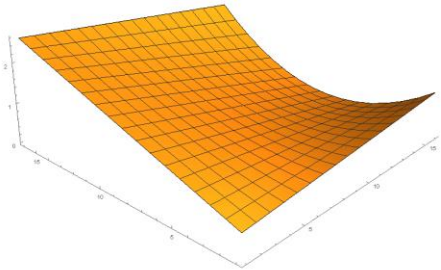
- Is it linear?
- Can it become linear?
  - Simple transformation: log/exp/sqrt/rcp
- Can it be divided into segments?
  - e.g. linear behavior in some segments
  - e.g. quadratic behavior in others



3. Create the approximate model

- Can we do dimensionality reduction
  - e.g. if the function is radially symmetric
- Does it sweep between two behaviors?
  - i.e. lerp(f(x),g(x),y)
  - Does it sweep non-linearly?





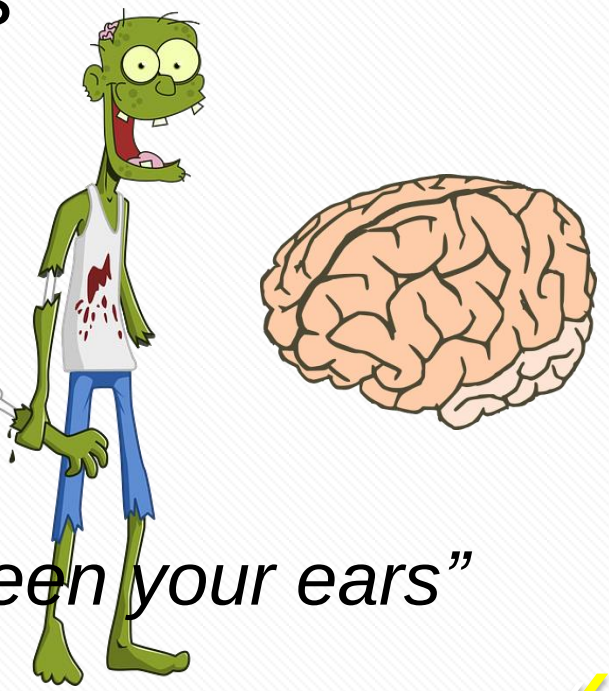
3. Create the approximate model

- If nothing simple comes out, try formulating the model in terms of different variables
  - Try combining terms
  - Try expressing some terms as combinations of others
- Careful with the number of free variables!

3. Create the approximate model

- Tools that can create models automagically exist :
  - DEAP, DataModeller, FFX
- But those are only tools
- To paraphrase Mike Abrash:

"the best modelling tool is between your ears"



4. Fit Parameters to your model

- Your model needs parameters
- Optimizer for every occasion:
  - Linear regression
  - Local:
    - Gradient descent
    - BFGS/L-BFGS
    - Levenberg-Marquardt
    - E-M
    - Nelder-Mead Simplex
  - Constrained/unconstrained
  - Global:
    - Differential evolution
    - Simulated annealing
    - Multistart

5. Check against ground truth

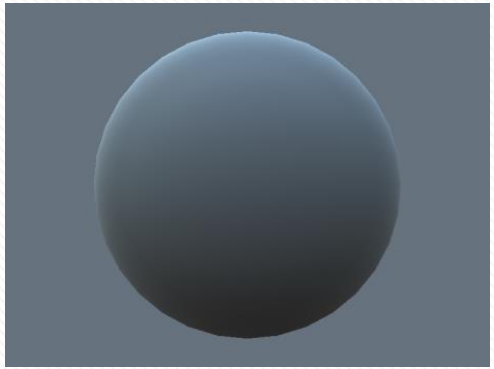
- Check the model in practice, not just on the plots!
- Lowest error doesn’t always mean the best result
  - Finding the right metric might be tricky itself!
- Flip between your approximation and reference
- Automate as much as possible
- Collect all results
- Ensure they are reproducible

Time for practice!

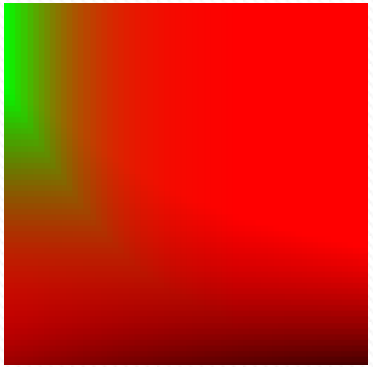
There’s more in the course notes!

Case study

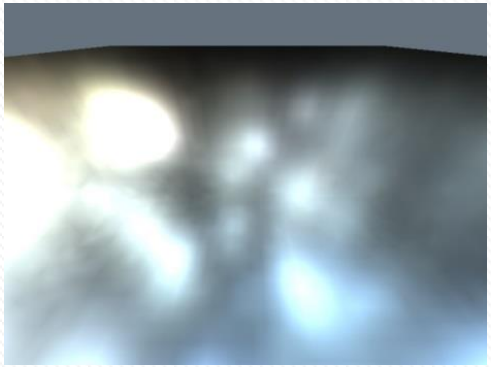
- Image based specular lighting with GGX-based microfacet model
- Most typical approximation:



Preconvolved cubemap



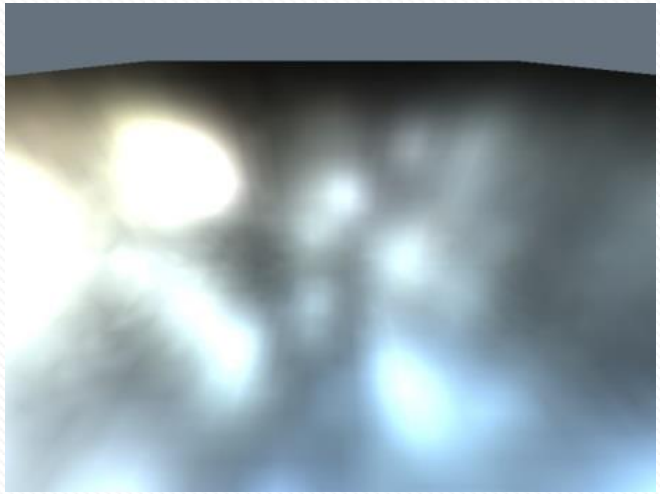
Split integral approximation



Single lookup at runtime

Issues with the current approach

- Preconvolving with symmetric kernel = no elongated highlights



!=

Step 1: Acquire the data

- The specular BRDF:

$$f_s = \frac{D(\theta_h)F(\theta_d)G(\theta_l,\theta_v)}{4\cos\theta_l\cos\theta_v}$$

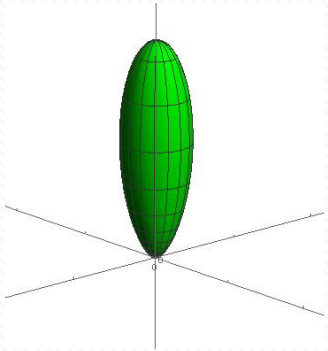
- Assume isotropic version and fixed F_0
- Combine the BRDF with the cosine term

$$L_o(\omega_o) = \int_\Omega L_i(\omega_i) f(\omega_i,\omega_o)\cos(\omega_i) d\omega_i$$

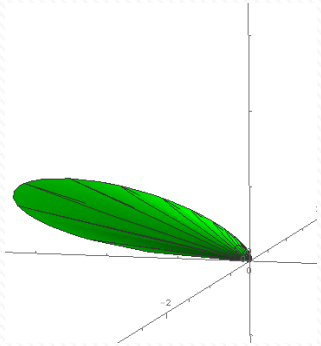
- Find approximation for arbitrary viewing angle and roughness

Step 2: Visualize

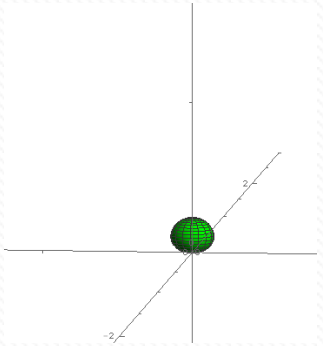
- Polar plots for different roughness values and viewing angles
- Shows amount of light from given direction contributing to result
- The lobe close to the reflected view vector
- Vertically stretched at grazing angles



Roughness 0.5, view angle 0



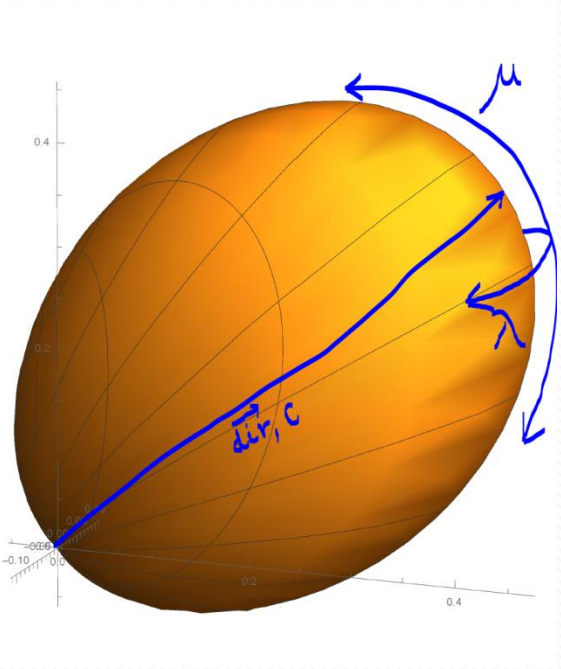
Roughness 0.5, view angle 1.57



Roughness 1.0, view angle 1.57

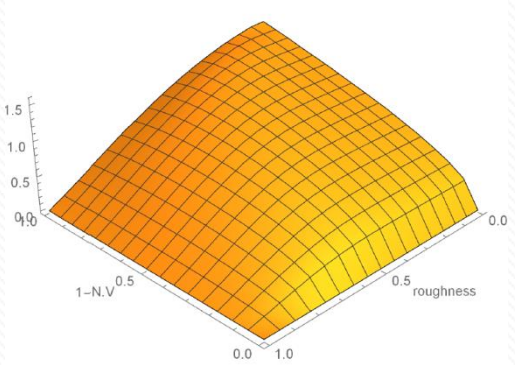
Step 3: Create the model

- Idea 1: Anisotropic Spherical Gaussian
  - Like Spherical Gaussian, just anisotropic
  - Just a handful of parameters to describe it

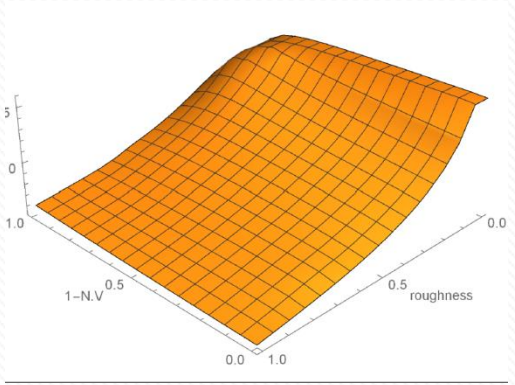


Step 3: Create the model

- The plan:
  - Uniformly sample the domain
  - Fit the ASG to the specular lobe for each point

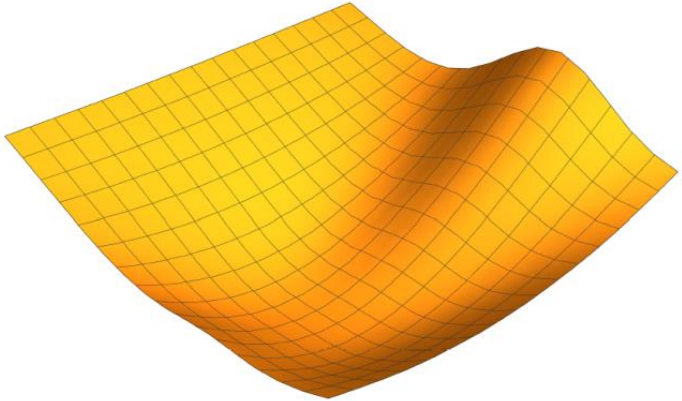


- At runtime:
  - Interpolate to get ASG parameters for actual view angle and roughness
  - Use hardware anisotropic filtering to lookup into prefiltered cubemap



Step 3: Problems

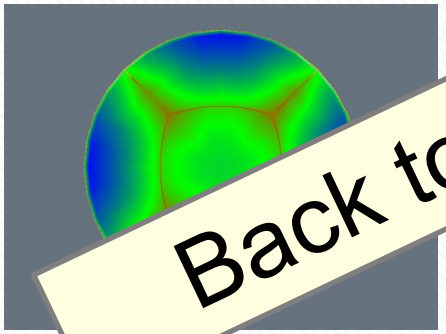
- Problem 1:
  - How to compute the error?
- Solution:
  - Just an integral:
    - Monte Carlo with importance sampling
  - Derivatives are costly:
    - Use Nelder-Mead that doesn't need them
  - Noise free with enough samples (30k-50k)



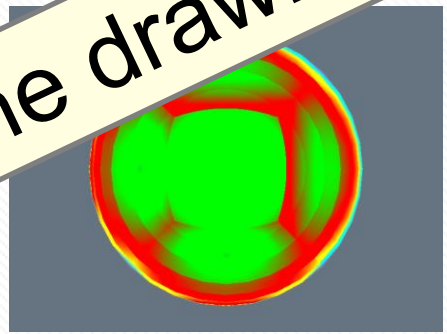
Step 3: More problems

- Problem 2:
  - anisotropic lookup on cu… …ar from reliable…

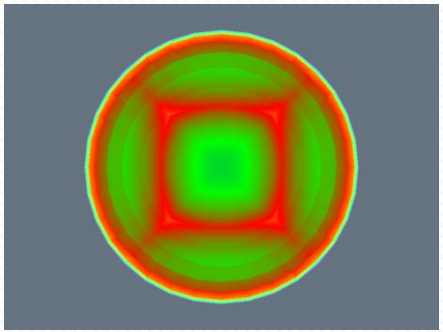
Back to the drawing board…



NVIDIA



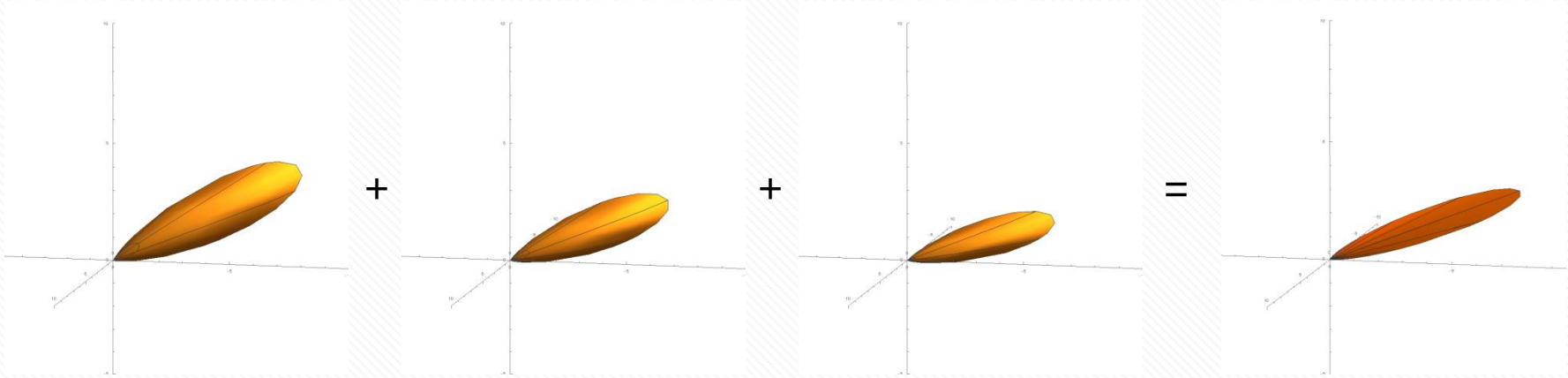
AMD



Reference rasterizer

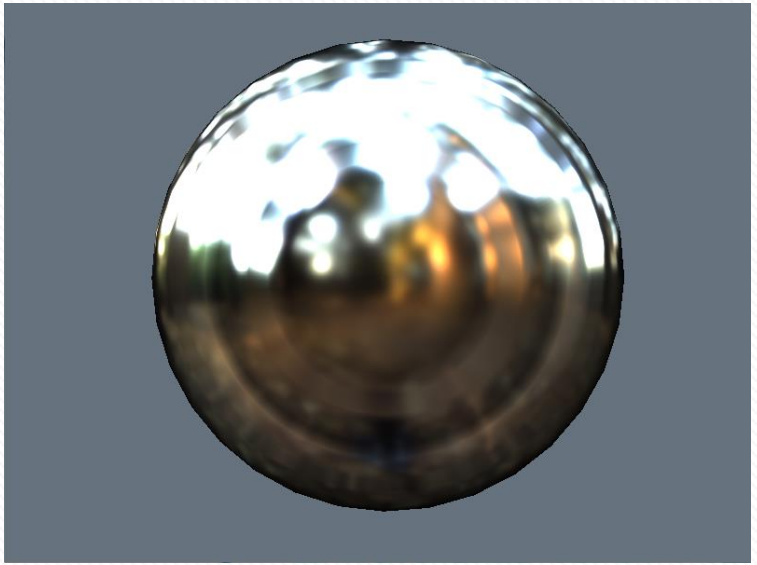
Step 3: Back from the drawing board

- Idea 2: Multiple Gaussian lobes, take multiple samples from cubemap



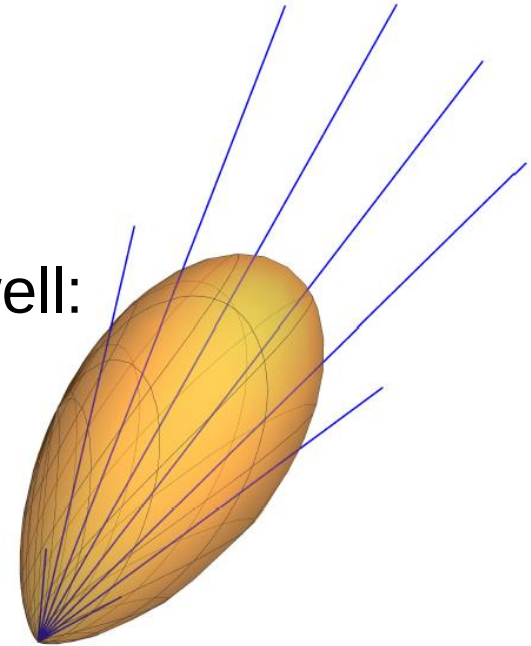
Step 3: Problems again

- Problems:
  - Each fit separate:
    - Lobes inconsistent
  - Whole domain at once:
    - Solver easily stuck in local minima



Step 3: Third time's the charm

- Combine the ideas!
  - Simple external model
  - Multiple lobes internally
- Handful of parameters that interpolate well:
  - Base direction
  - Spread
  - Bandwidth (shared)
  - Amplitudes follow Gaussian curve



Step 4: Fit the parameters

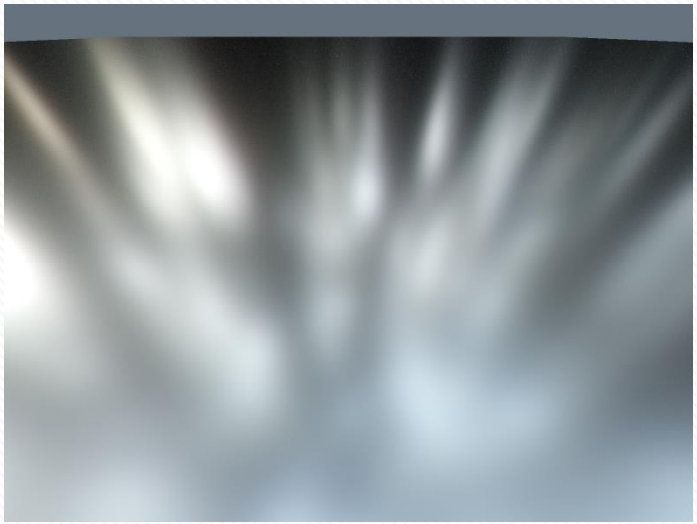
- Nelder-Mead fit for every view angle and roughness combination
- Local algorithm – it needs a good starting point:
  - Generate near the expected solution
  - Scan the domain and pick points with low error

Step 4: Fit the parameters

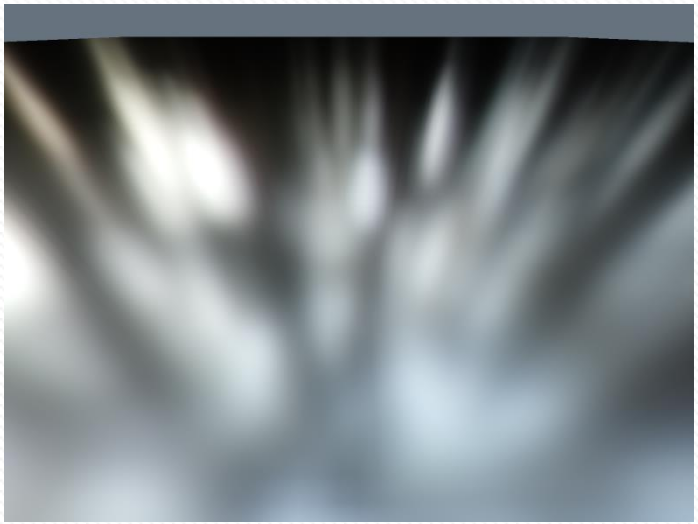
- Fit values for roughness x view angle domain
  - Each data point independent
  - Each starting point independent
    - Parallelize!
- Two variants:
  - Amplitude forced to generate equal energy as GGX
  - Amplitude as free variable

Step 5: Validate

- Put it all into a shader…
- Compare against the ground truth



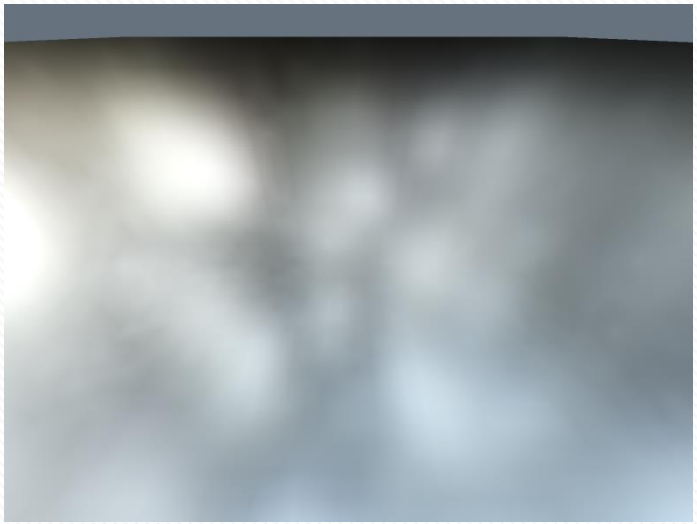
Reference (importance sampled)



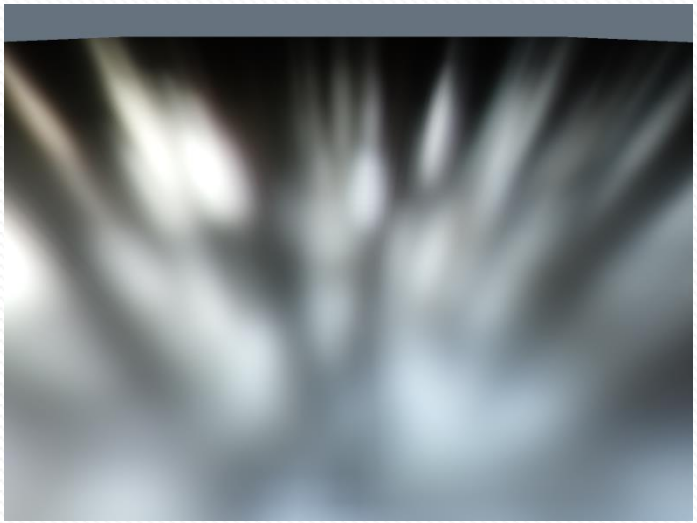
Described model

Step 5: Validate

- We’ve got the elongated highlights!



Old approximation



Described model

Step 4: Fit the parameters

Equal energy

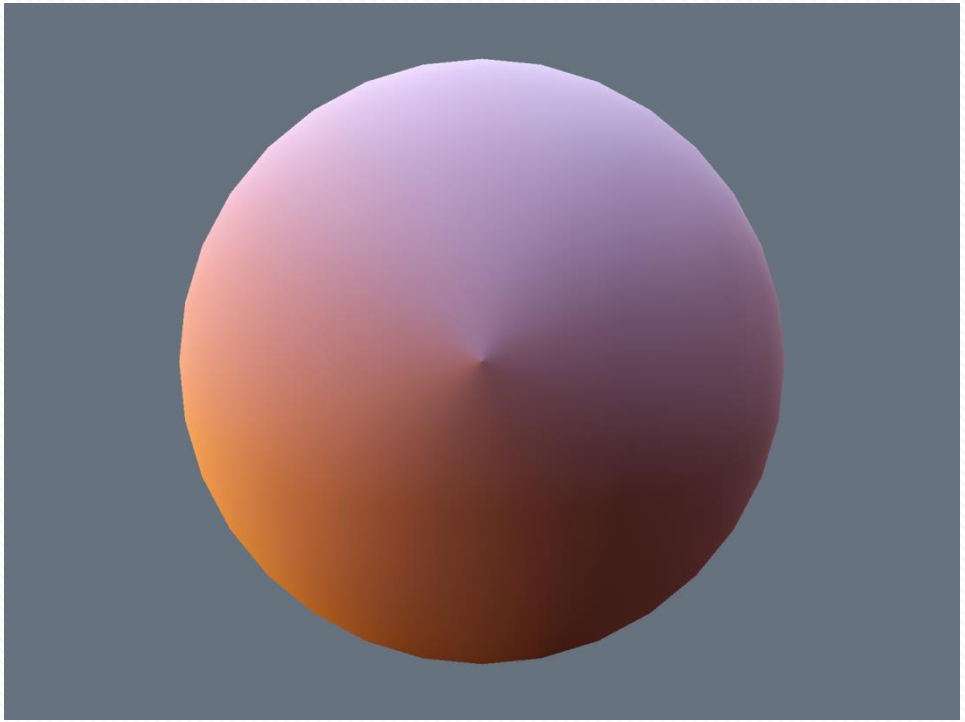
Step 4: Fit the parameters

Free amplitude

Step 5: Validate more

- What happens at high roughness?
- When spread > θ we get radial artifacts

- Damn, so close...

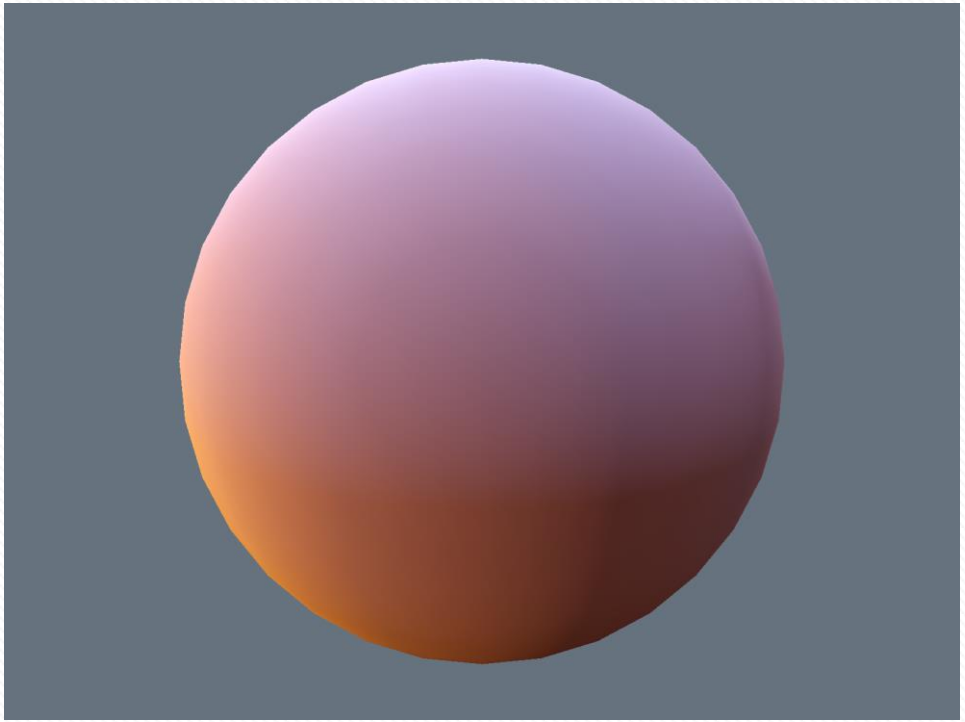


Corrections to the model

- Add a constraint, make sure that spread is no larger than θ
- Super simple with Nelder-Mead:
  - Penalty term
  - Trivial changes to the algorithm itself
- Not entirely kosher, but works just fine in practice

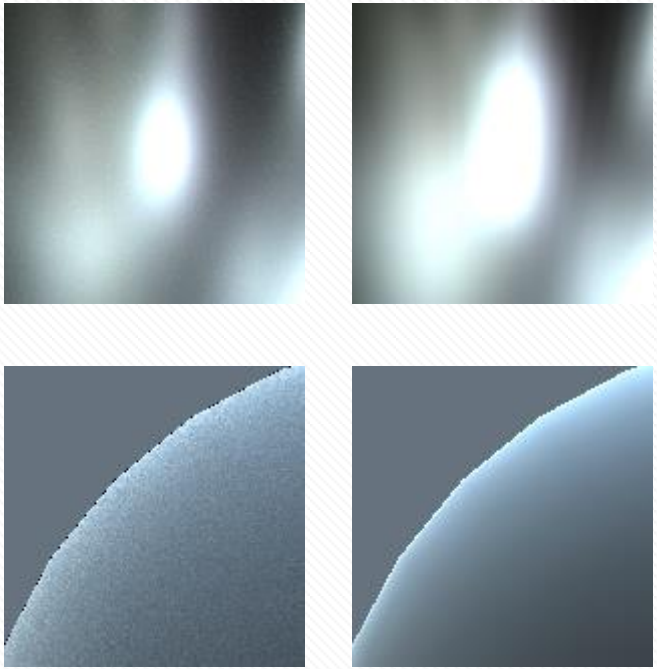
Step 5.1: Validate again

- Much better



Conclusions: issues

- Shorter tail than when using GGX
- Not very accurate at high roughness values
- Could be more flexible at grazing angles



Conclusions

- What's next?
  - Different number of lobes for different parts of the domain
  - Anisotropic BRDFs
  - Combine with frequency-based normal map filtering for shader antialiasing

More in the course notes!

Much, much more!

Area lighting for GGX specular

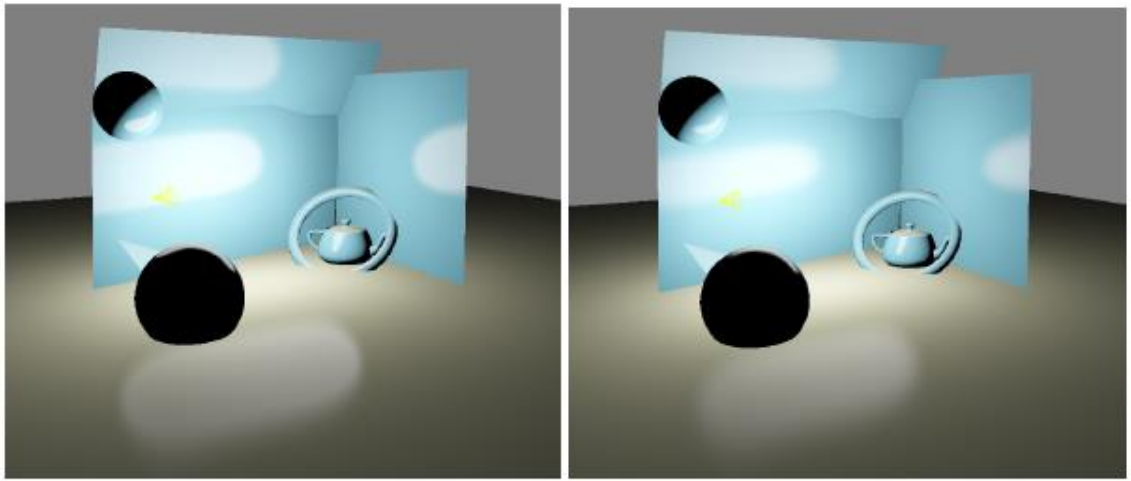


Figure 18: Capsule lights rendered with: on the left the original approximation by Karis, on the right the new smoothed representative point solution.

Analytical approximation for the split-integral IBL texture

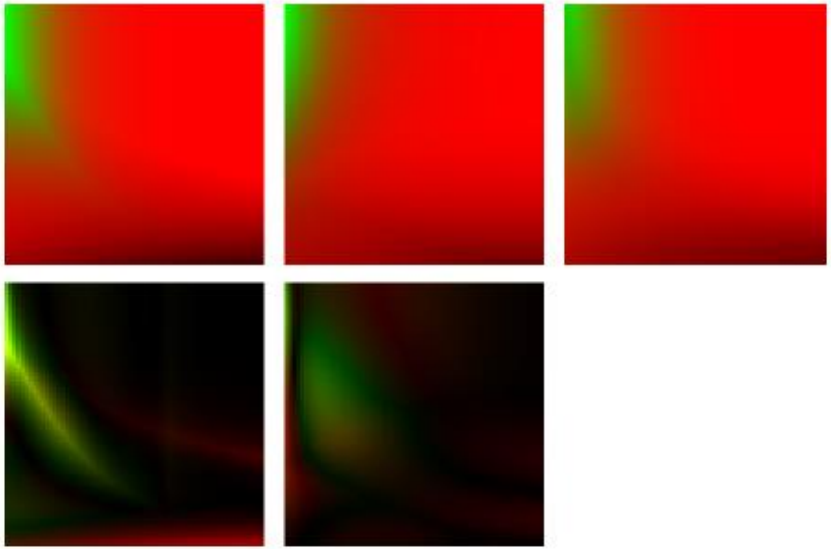


Figure 22: From left to right: Narkowicz formula, approximation via symbolic regression, original data. Bottom row shows absolute difference with the original data times 3.

Image-based lighting for the Disney Diffuse BRDF

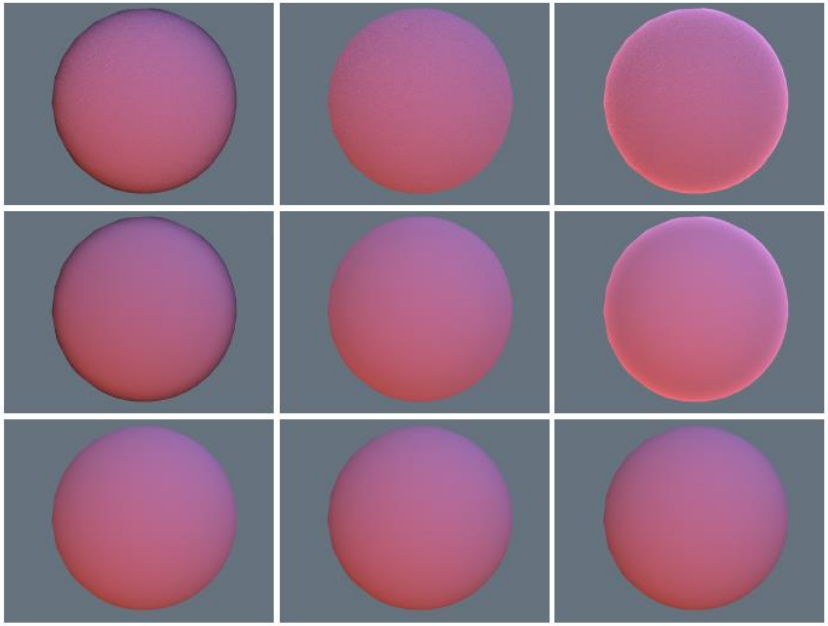
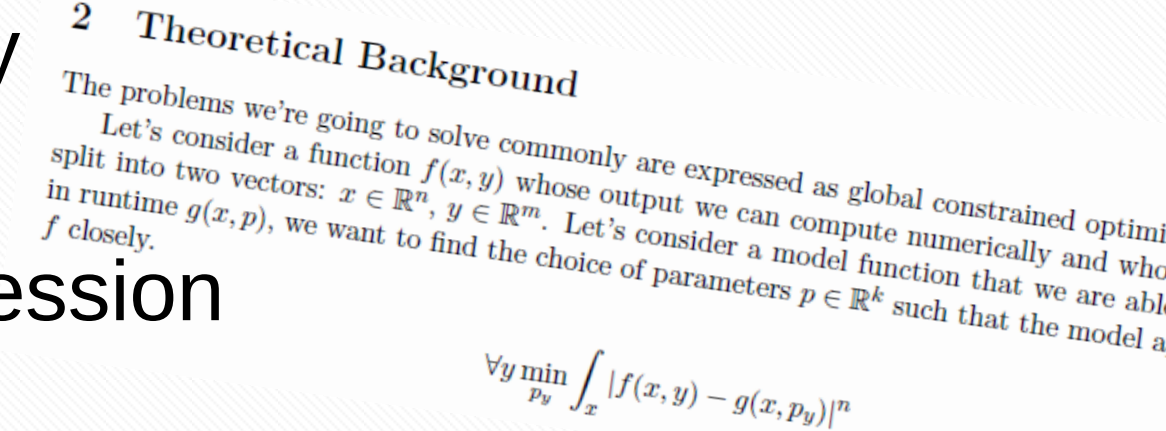
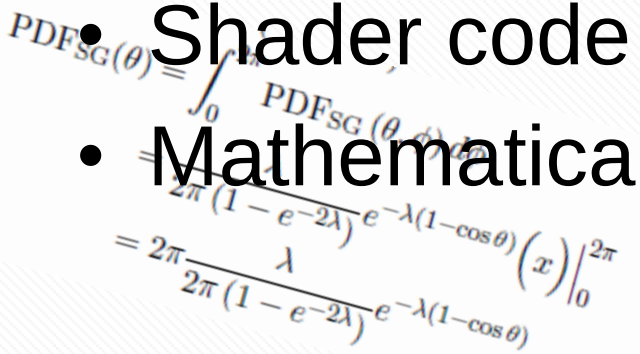


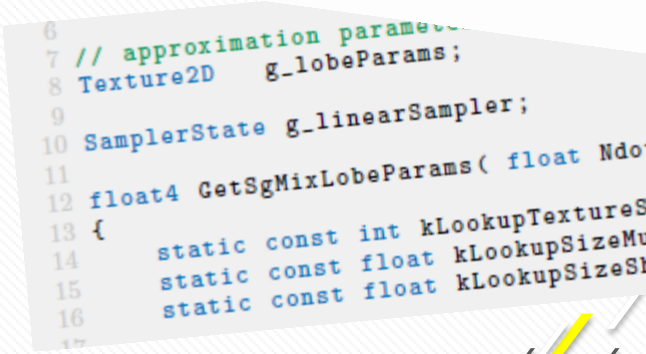
Figure 29: Comparison of the ground truth (importance sampled) version of the Disney Diffuse BRDF (top row), the proposed approximation (middle row) and the regular Lambertian model (bottom row) for different roughness values (left to right: 0.0, 0.5, 1.0).

And more!

- Bunch of theory
- Fancy math
- Symbolic Regression
- Shader code
- Mathematica notebooks







Acknowledgments

- Steve Hill, Steve McAuley
- All the smart people we work with, including but not limited to:
  - Peter-Pike Sloan
  - Joe Manson
  - Mike Stark
  - Paul Edelstein
  - Jorge Jimenez
  - Danny Chan
  - Dave Blizard
  - Demetrius Leal
  - Dimitar Lazarov

Q&A

Contact us (we mean it):

michal.iwanicki@activision.com

angelo.pesce@activision.com, @kenpex