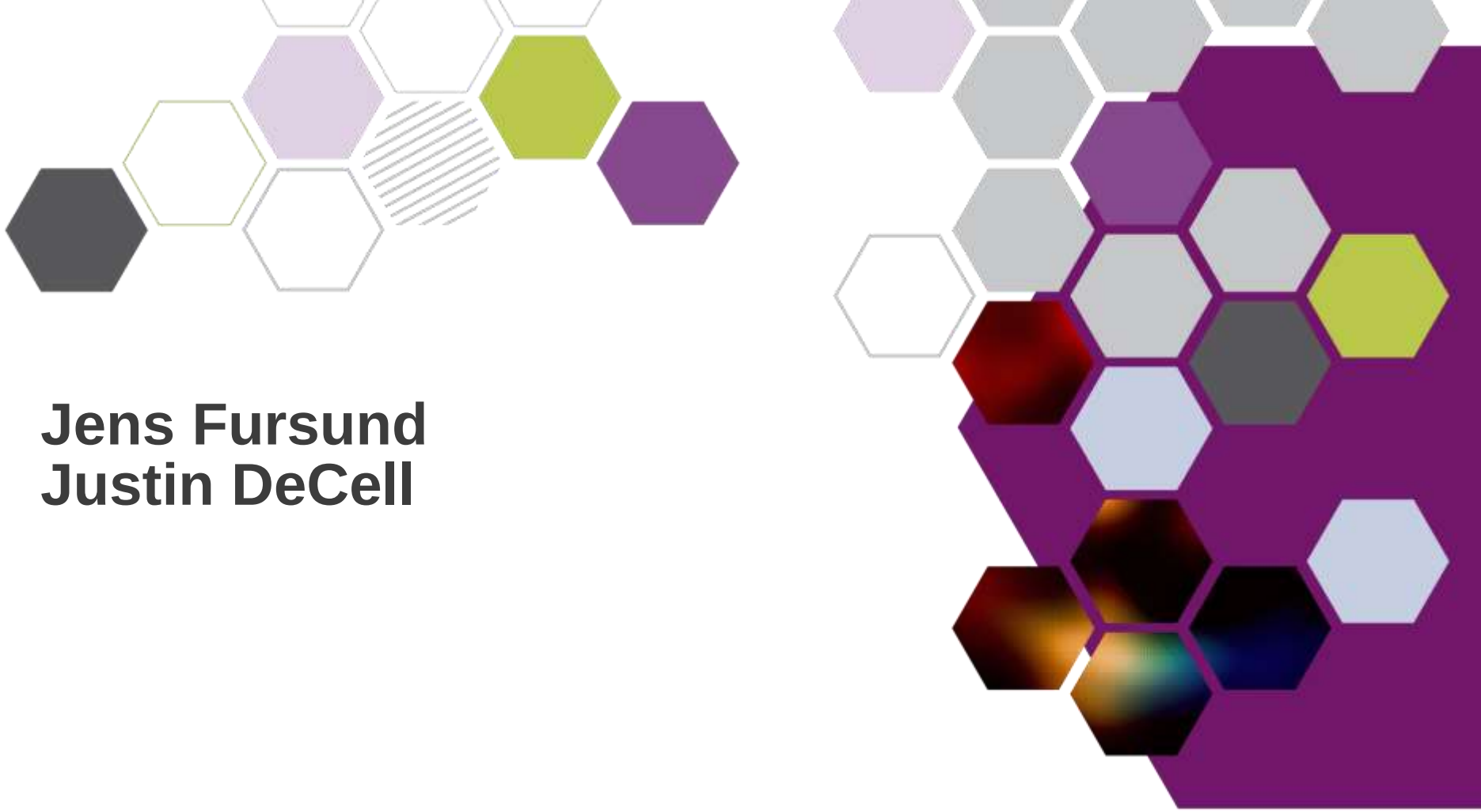
idc16

Imagination Developers Connection

Interactive Light Mapping with PowerVR Ray Tracing

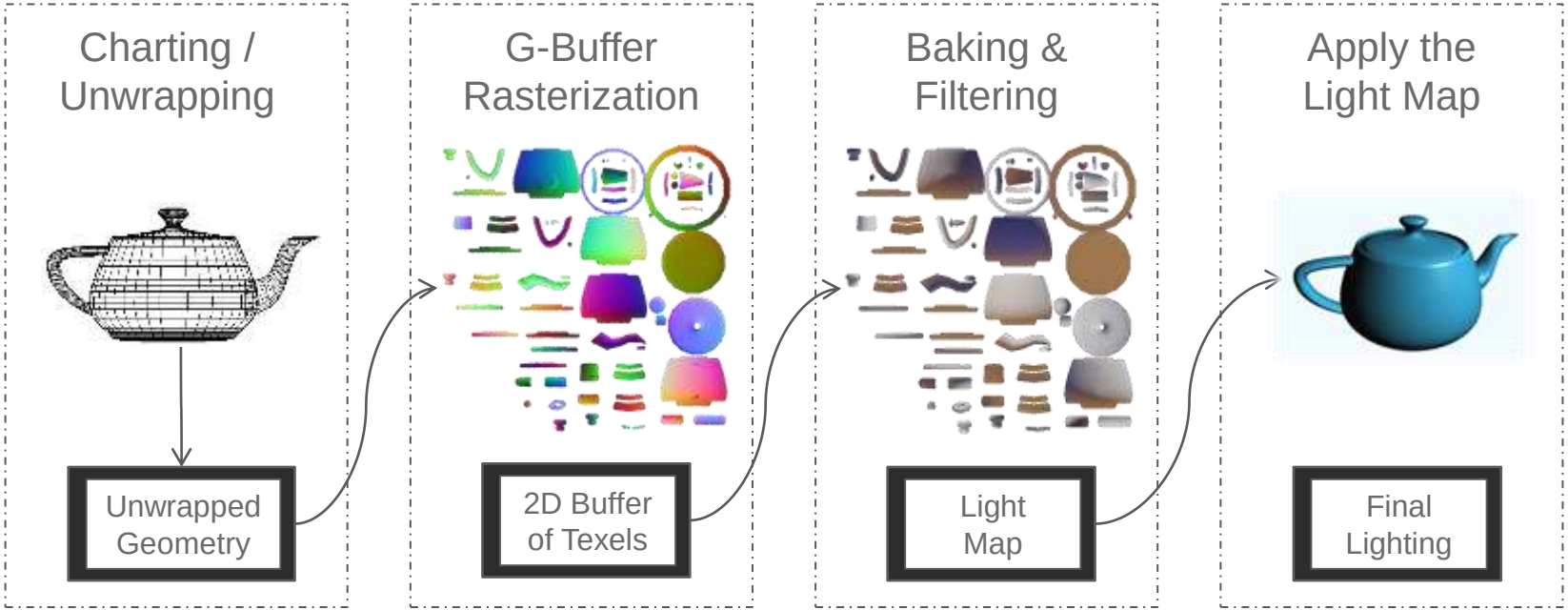


Jens Fursund
Justin DeCell

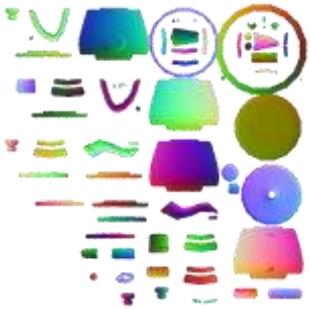
Light Map Basics

A light map is a texture that stores lighting for objects in the scene

Generation of light maps for GI



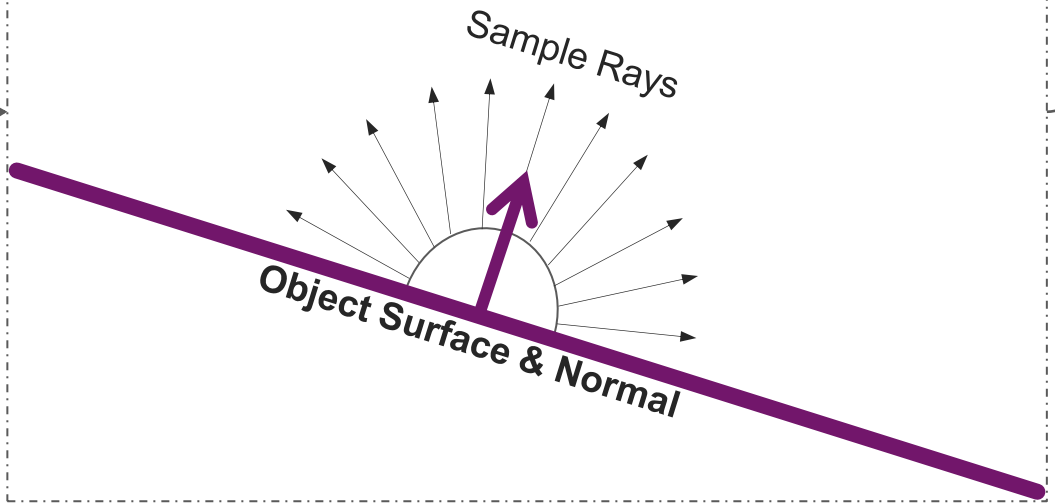
Oven Fresh Lighting



2D Buffer of Texels

Baking

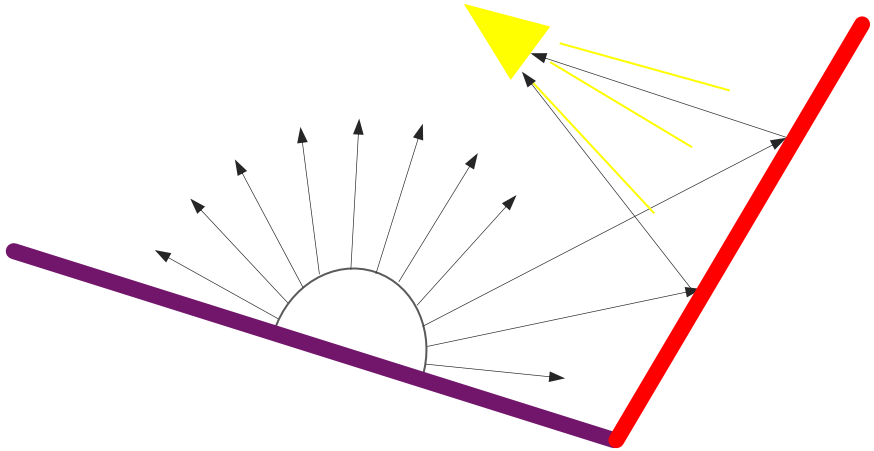
Render the lighting for each texel represented in the g-buffer by stochastically (randomly) casting rays into the scene from the point in space saved in the g-buffer



Light Map

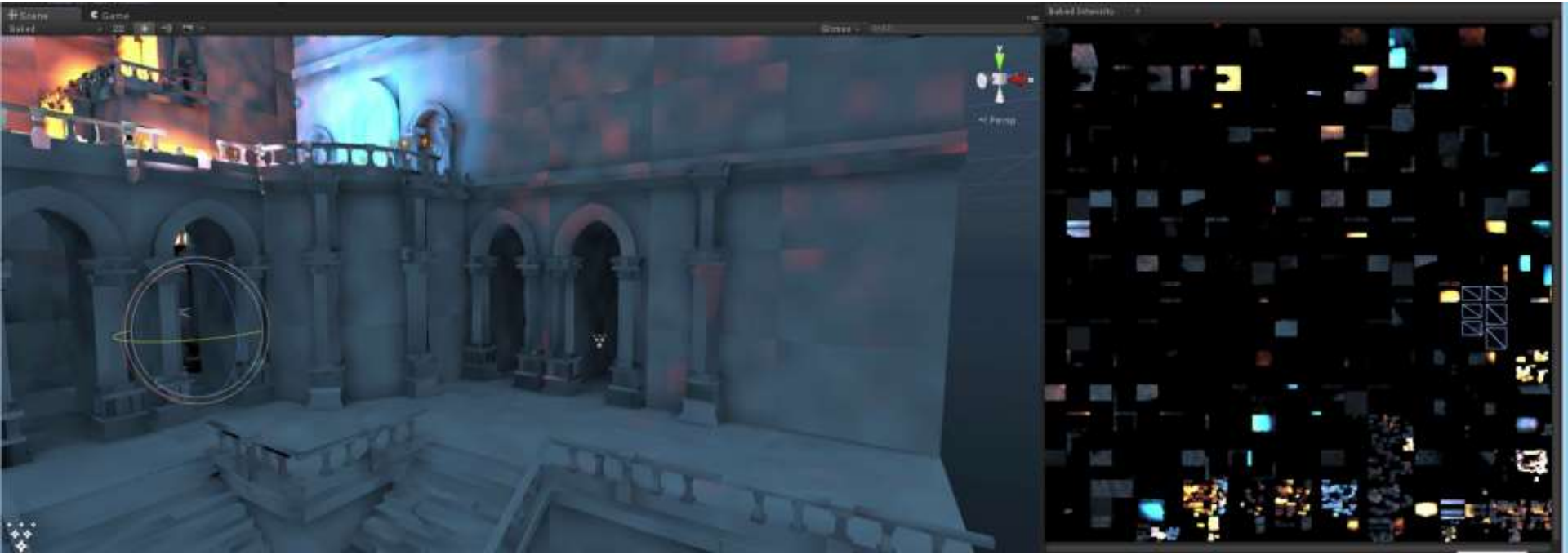
Converging…

- To get smooth lighting we need to evaluate all the light coming on to a point
- Practically impossible, so we'll try to get close
- This is what we call converging on the final result



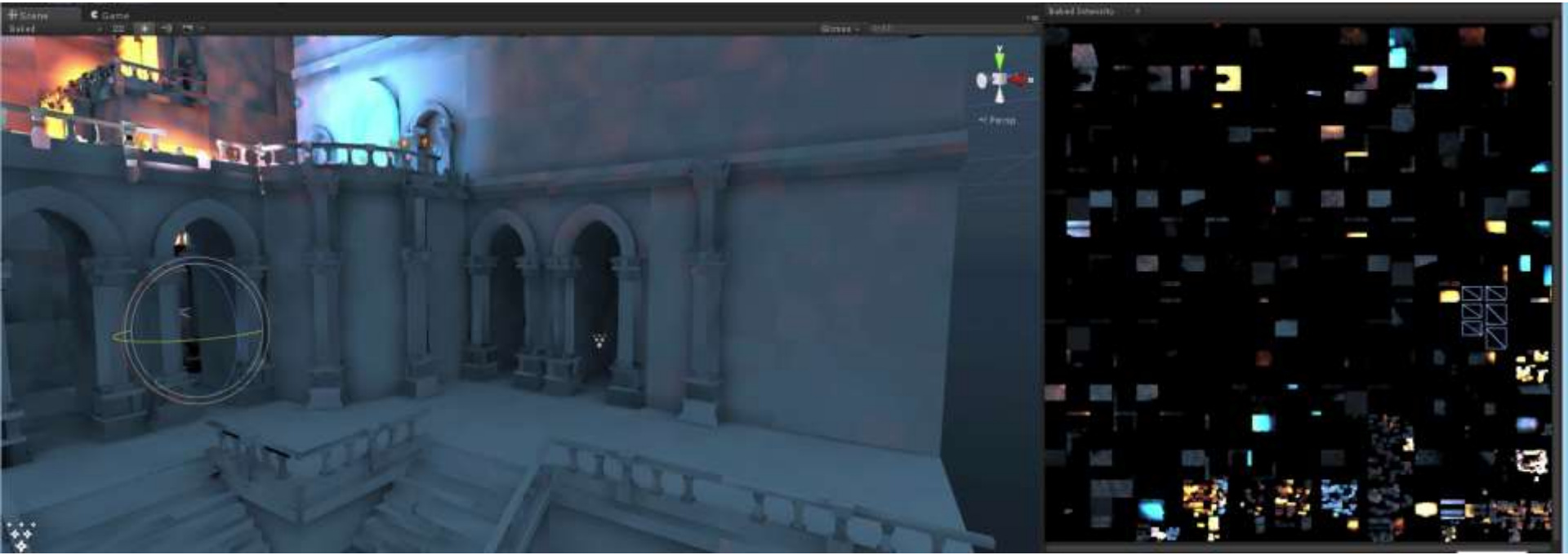
Progressive Refinement

Just a few rays means the result is very noisy



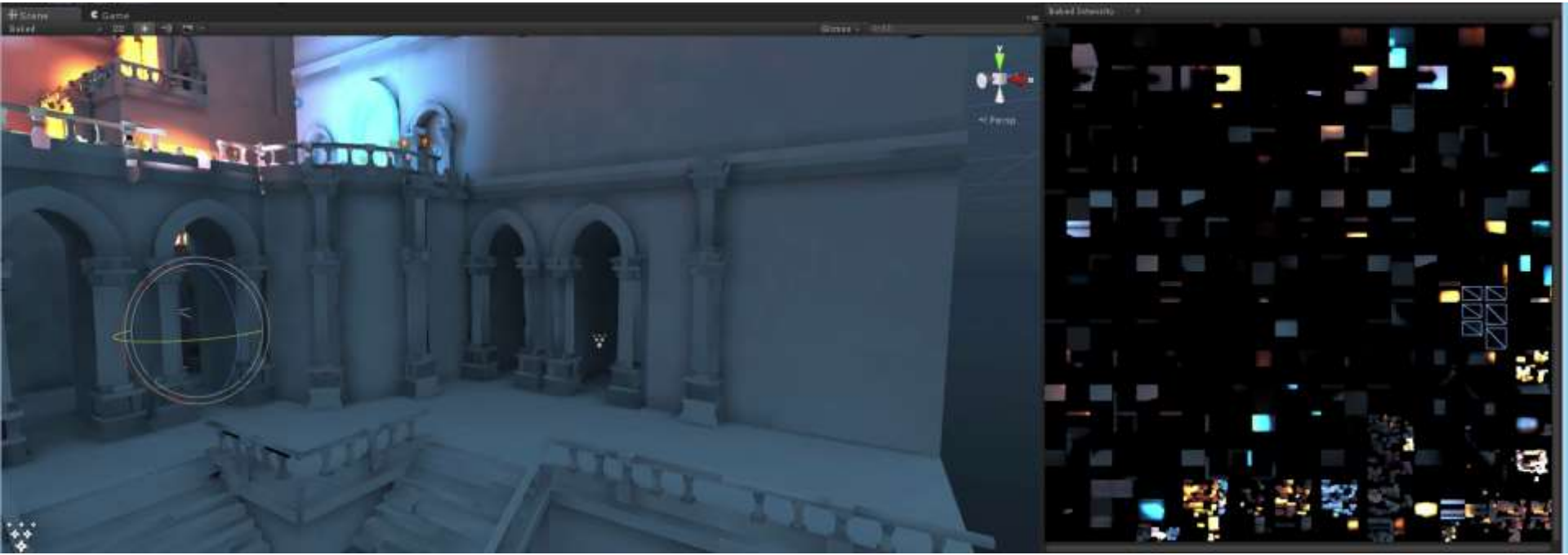
Progressive Refinement

Over time, more rays are averaged into the data reducing noise



Progressive Refinement

With enough rays, the light map looks perfect



Demo

PowerVR Light Mapping in Unity 5

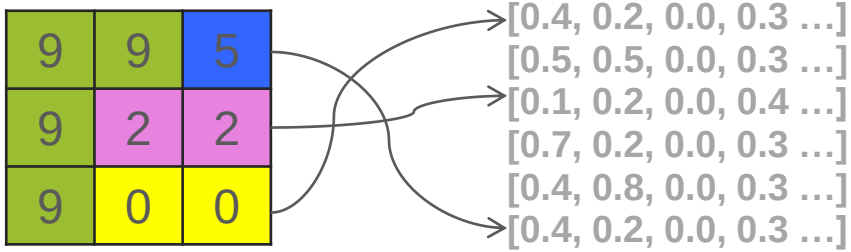
What about dynamic worlds?

Progressive baking can happen during game play

- **Baking during game play can get us dynamic worlds**
- **We achieve this by:**
  - Object space g-buffer to rasterize the g-buffer less often
  - Use the light maps as a light cache to save on rays
  - Amortize the cost over several frames

Object space g-buffer

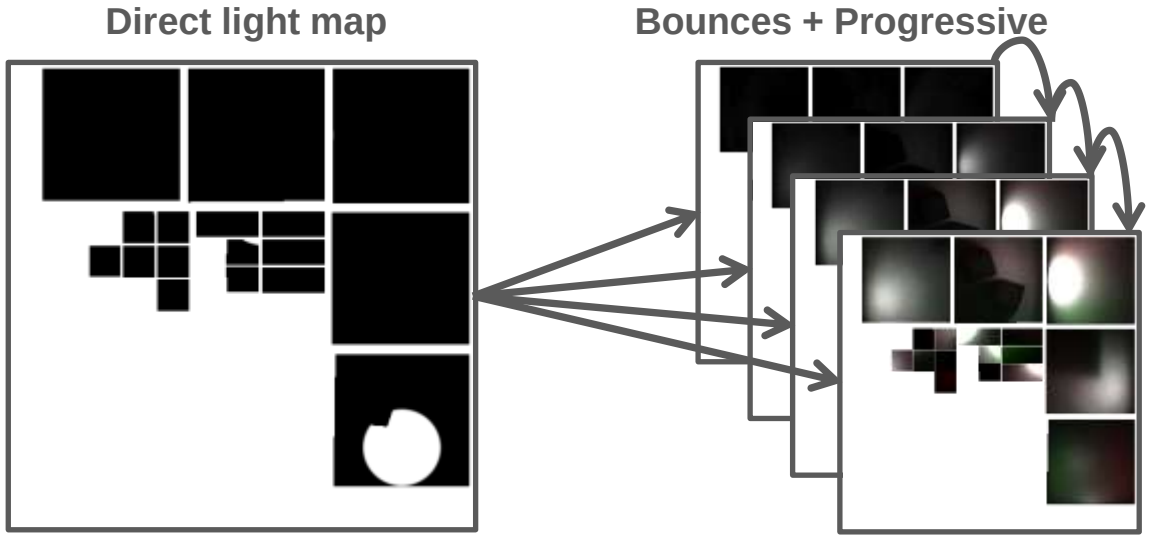
- **Rasterize the uv-space g-buffer in object space**
  - Store object space normal and position
- **Store an id to a world-space transform array per texel**
  - Use the id to look-up into the world-space transform array
  - Update the transform array when an object moves



Light maps as a light cache

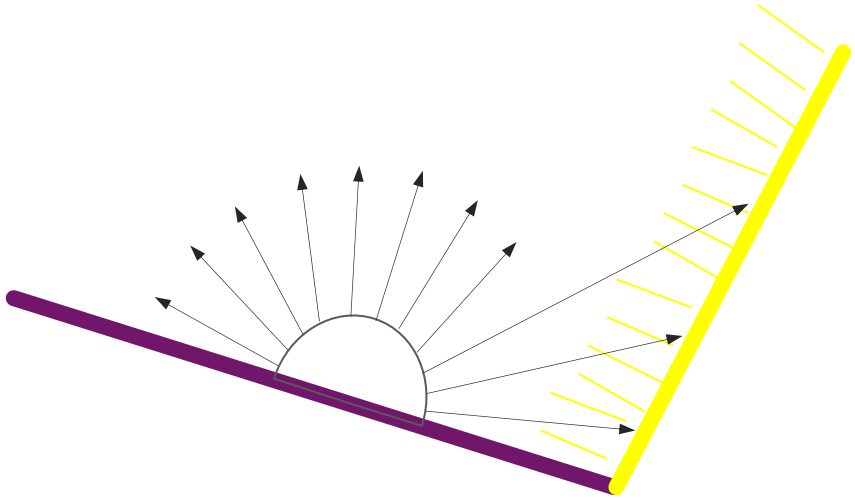
Dynamic light maps are basically a cache for lighting

- **When a ray hits an object, lighting is fetched from the light map**
  - This saves the work (and rays) of evaluating the lighting where rays hit
- **Current baking pass is used as bounce in the next pass**



Emissive is free!

- Traditionally evaluating emissive surfaces is computationally expensive
- Here we can treat it as any other light cache, so it comes for free!



Amortize the cost over several frames

Progressive baking can happen during game play

- **Progressively accumulate results over several frames**
- **Average in new results continually if the scene changes**

Amortize the cost over several frames

Progressive baking can happen during game play

- **The speed of scene changes dictates the amount of time that the game can take to rebake the lighting**
  - Demonstration done by [Crassin et al, 2013] demonstrates people don't notice lags in indirect lighting for at least 500ms. Slower direct light changes make it even longer
  - We call this the tolerable_latency

Amortize the cost over several frames

Progressive baking can happen during game play

- **Other factors effecting bake latency include**
  - The rate at which the hardware can trace rays
  - The number of texels in the light map you are baking
  - The time you can give to the light map baking process, each frame
  - A constant “decay_threshold” that represents the amount of the old light map that can exist in the new light map after tolerable_latency. We use 10%

The running average formulas

- Previous GI Dilution Factor

$$e^{\ln(decay\_threshold)/tolerable\_latency*frame\_rate}$$

- Current GI Weight

$$pass\_weight = 1 - dilution\_factor$$

- Can calculate the samples per texel using a target baking time

$$hardware\_ray\_rate * time\_for\_baking\_pass/num\_lightmap\_texels$$

Demo

In-Game Baking on Wizard

Sometimes you need loaded dice

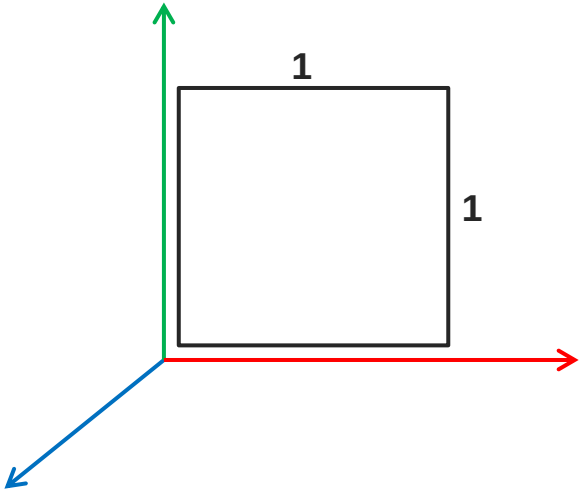
When your luck runs thin in Monte Carlo

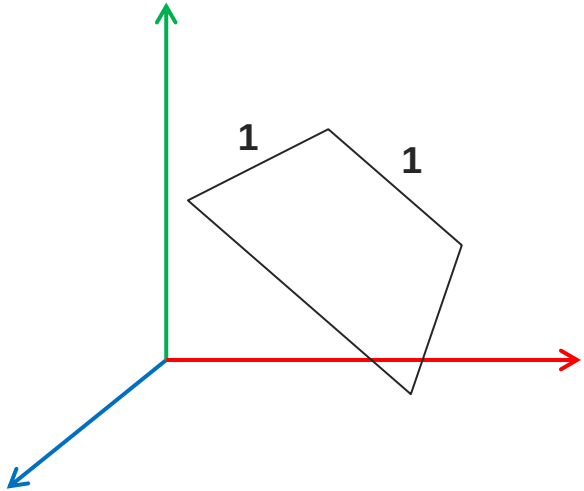
- **Stochastic Monte Carlo sampling is great if there is a large area of the scene contributing lighting**
  - It falls flat on its face when all of the light is coming from a small area like an open doorway or a spotlight shining on a wall

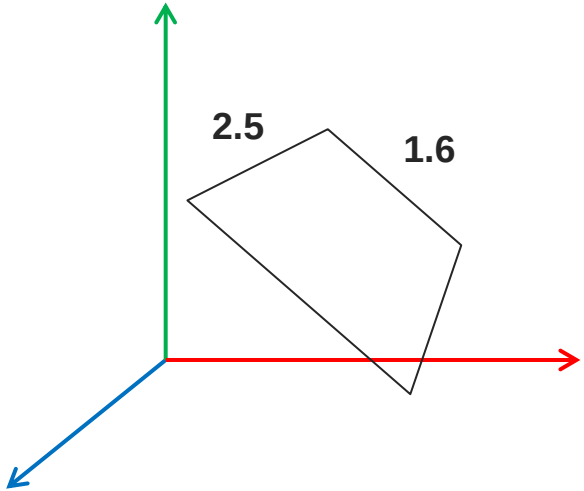
- **Importance sampling lets you aim your rays at the areas likely to contribute the most to the lighting**
  - We start with a rendering of the direct lighting into a version of the light map
  - We choose texel luminance multiplied by projected area as our importance value

Start with an energy-conserving light map

- **To base importance on luminance, we need to be sure we have the correct world space luminance for each texel**
  - Multiply luminance of each texel by world space area of texel



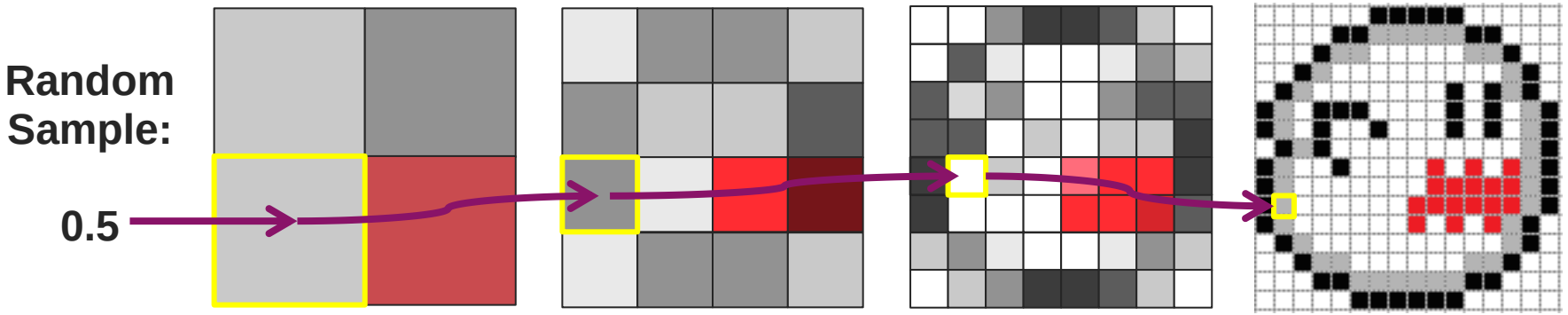




Build a Cumulative Distribution Function

Or a Mip Map will do ☺

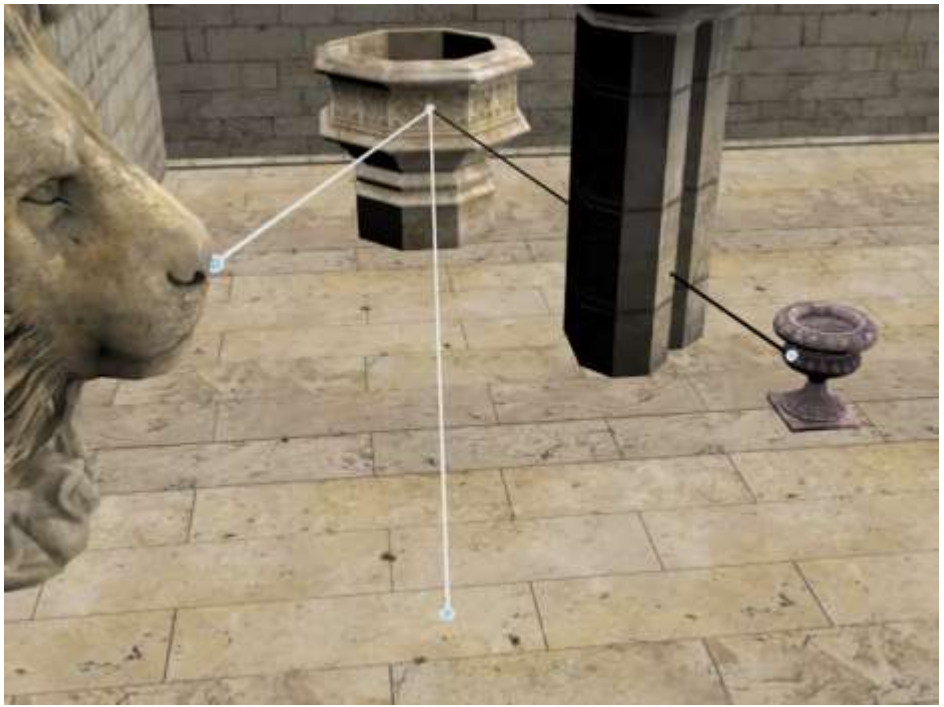
- **The direct light map luminance, aka the "source map", is a 2D probability distribution function**
- **A table that sums all of the values is called a cumulative distribution function. Averages are a type of sum**
  - The CDF can be traversed as a tree



Sample our CDF

Use Rays as Line-of-Sight Queries

- **To bake, each sample randomly selects a texel in the direct light map, and uses a ray to determine if there is line of sight from the source texel's position to the destination texel's position**



The Math

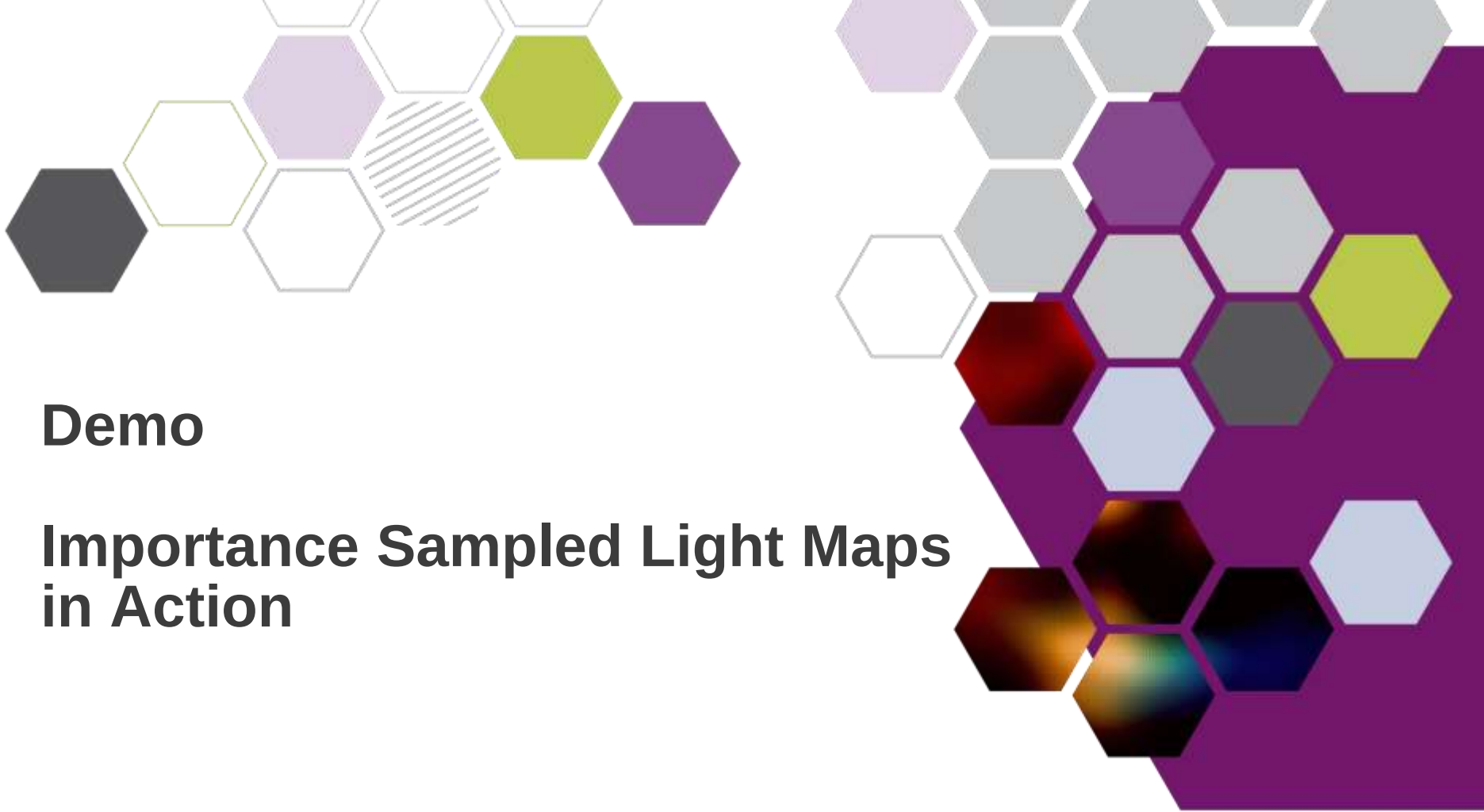
It doesn't have to be boring. But it probably is…

- **We must match the light contribution from a known source to the chances of finding it with stochastic sampling using geometric equations. Energy Conserving Light Map for the WIN!**

$$ray\_contribution = src\_texel\_value * src\_texel\_area * dot(ray\_direction, src\_normal) * dot(-ray\_direction, dst\_normal)$$

- **We must also weight each sample with the inverse of the importance value to normalize the contribution**

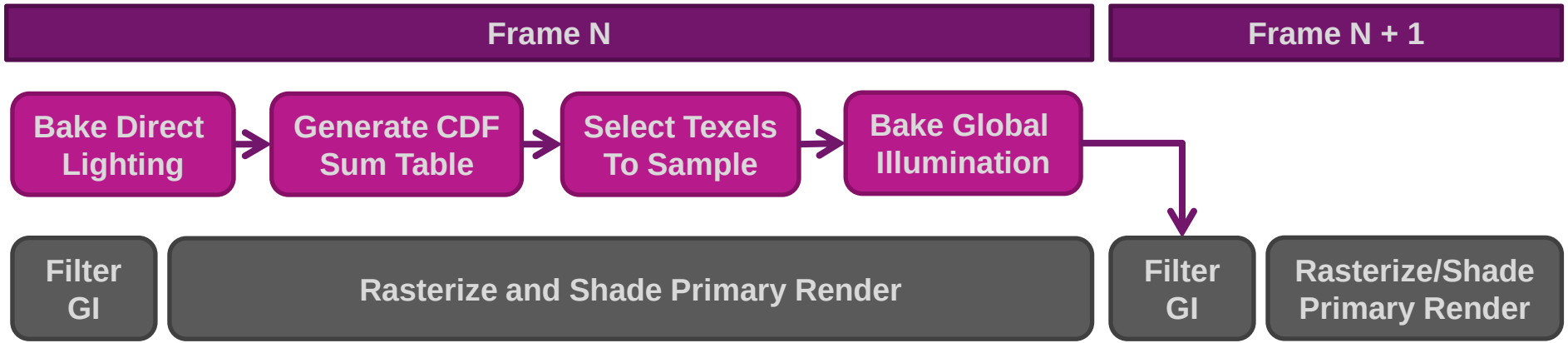
$$total\_luminance / (sample\_luminance * samples\_taken)$$



Demo

Importance Sampled Light Maps in Action

Rendering Pipeline



Current performance

- **512x512, 32 rays per texel per frame**
- **Total light map bake time: 32 ms (on a smartphone class Wizard GPU)**
  - Overlaps completely with camera-space rasterization and shadow maps
  - Less workload when the scene or direct lighting isn't changing
    - No direct light map baking
    - No building of CDF
    - We can increase rays per texel to get faster convergence

Future work

Lots more can be done to improve quality and speed

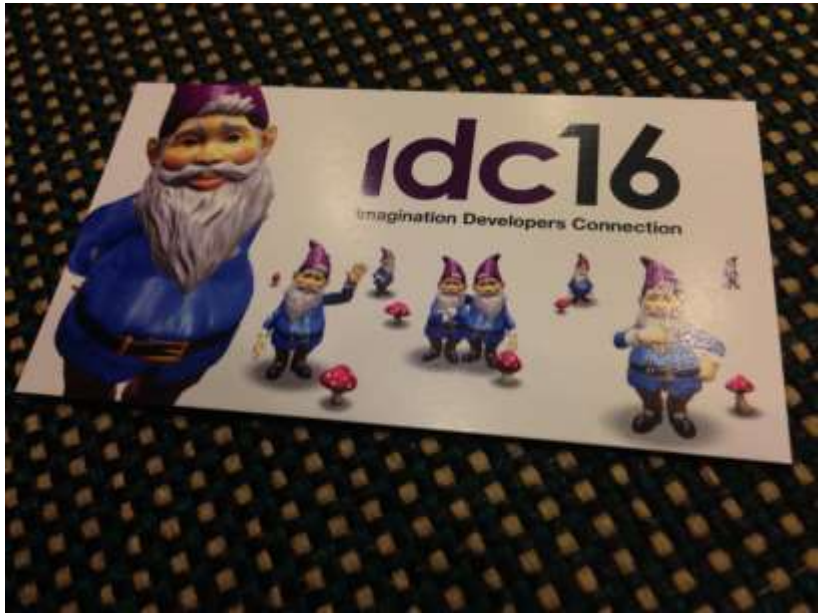
- **Visibility prioritization**
  - PowerVR Light Mapping in Unity already does this
- **Directionalized data in the light map**
  - Light map texels can carry coefficients for a spherical harmonic function
- **Importance based on distance**
  - Faster refinement with the same number of rays

Questions?

Get in touch:

- *Jens.Fursund@imgtec.com*, *Justin.DeCell@imgtec.com*
- ***@jensfursund***

Please come and visit us at booth #1902 in the South Hall to see our demos and to collect your very own Vulkan™ Gnome t-shirt!



Up Next…

Advanced Techniques for Ray Tracing