Rendering Tutorial I - Introduction

Rui Wang

rwang@cad.zju.edu.cn

State Key Lab of CAD&CG, Zhejiang University

2018/9/20

Syllabus

- Rendering Tutorial I: Introduction
  - Rui Wang, Zhejiang University
- Rendering Tutorial II: Monte Carlo Path Tracing
  - Shuang Zhao, University of California, Irvine
- Rendering Tutorial III: Materials
  - Lingqi Yan, University of California, Santa Barbara
- Rendering Tutorial IV: Precomputed Radiance Transfer
  - Kun Xu, Tsinghua University

Rendering Tutorial I:
- Introduction

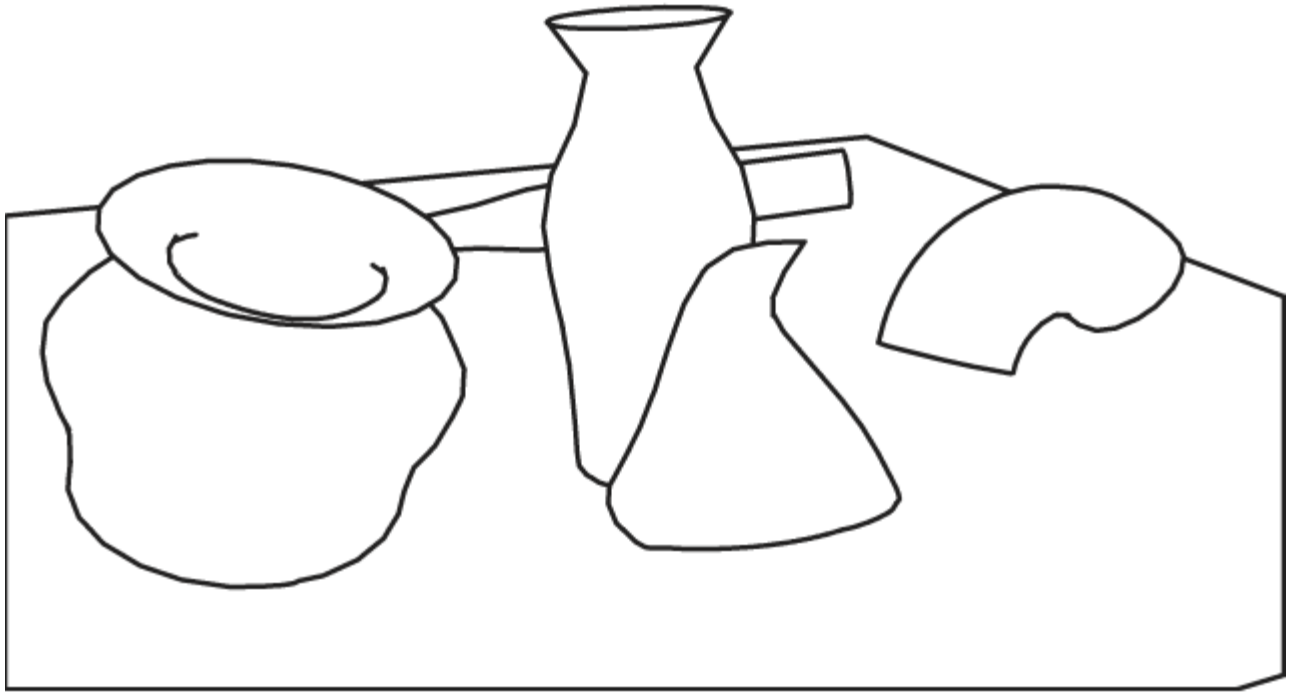
Abstract

Rendering（渲染/绘制）是图形学研究的重要方向，也是当前诸多应用领域（例如：游戏、电影、VR/AR等）的重要基础技术，具有广泛的应用背景。本报告作为一个4节Tutorial系列的第一个报告，将首先简要介绍当前绘制研究的现状，其次介绍绘制技术相关的基础概念与知识，主要内容包括：绘制方程简介；求解绘制方程的两大类基本思路——真实感绘制与实时绘制，以及由不同思路导致的不同方法与技术体系。最后将对绘制技术的发展做一些展望。

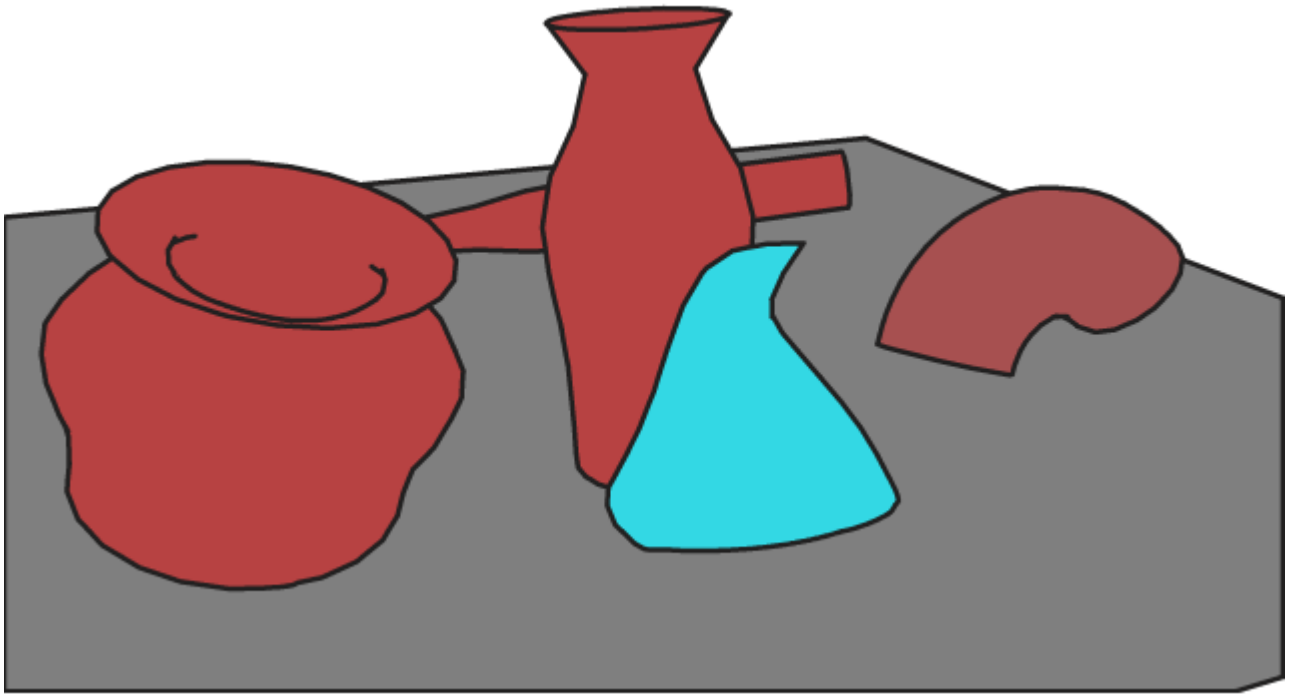
http://www.cad.zju.edu.cn/home/rwang

Rendering

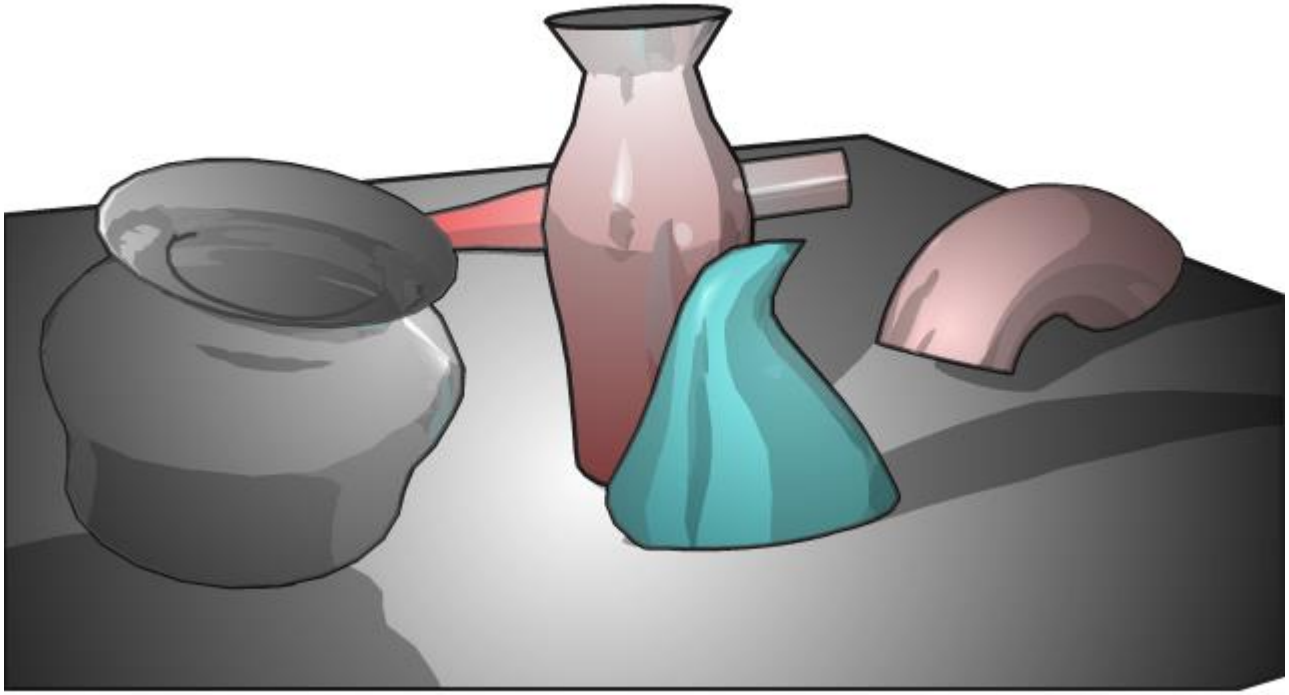
Rendering



Rendering



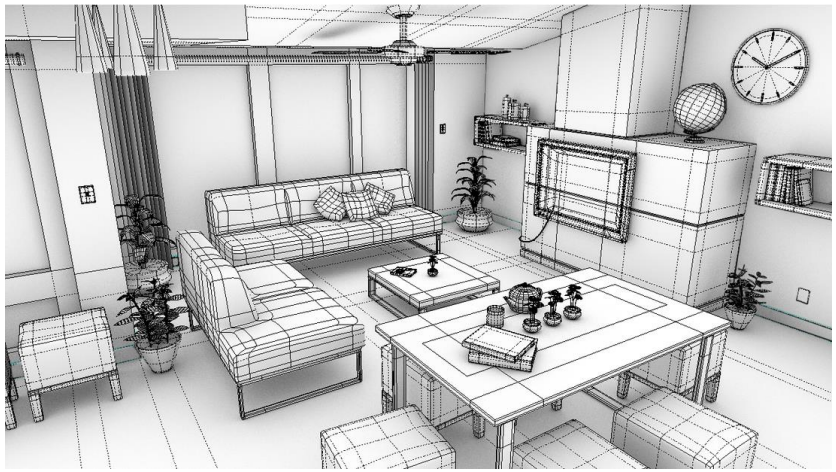
Rendering



Rendering

Rendering

Rendering



- A key interface between Human and Computer
- A fundamental technique of CG, VR, Digital Entertainment and Vis.
- It has been widely used in many industrial applications

Game

Movie

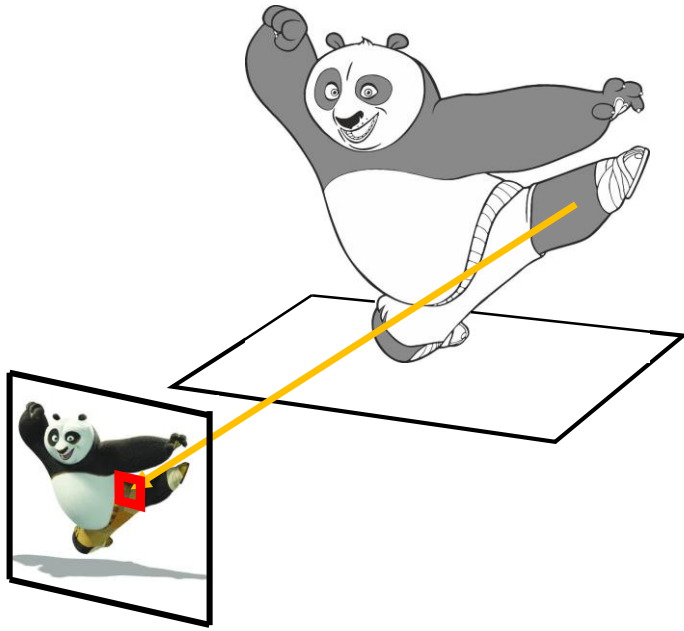
Virtual/Augmented Reality

Virtual World in Ready Player One

How to achieve it?

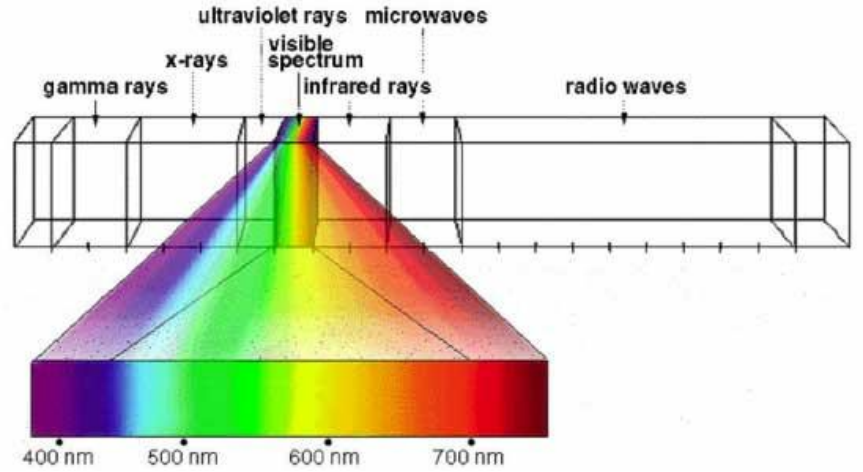
To The Rendering Equation

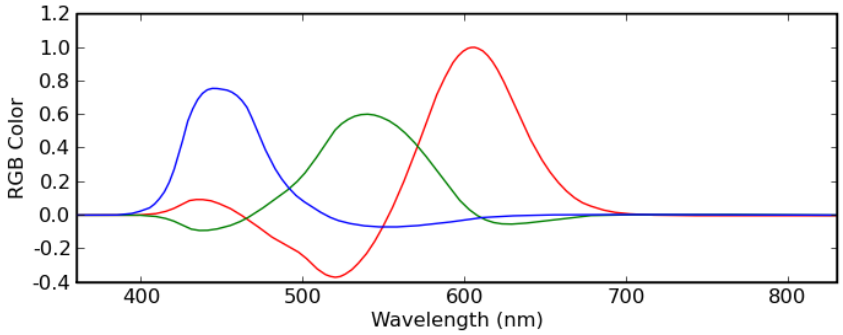
- What is the **color** of pixel (i,j)?

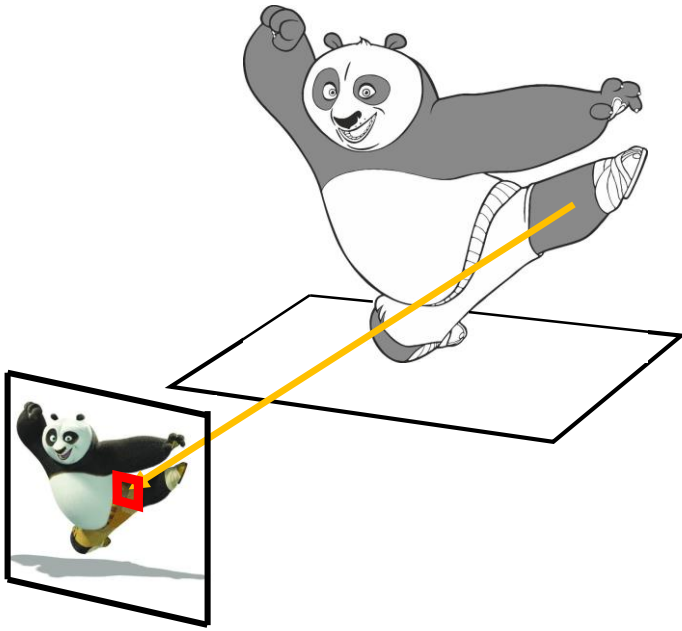


To The Rendering Equation

- What is the **color** of pixel (i,j)?





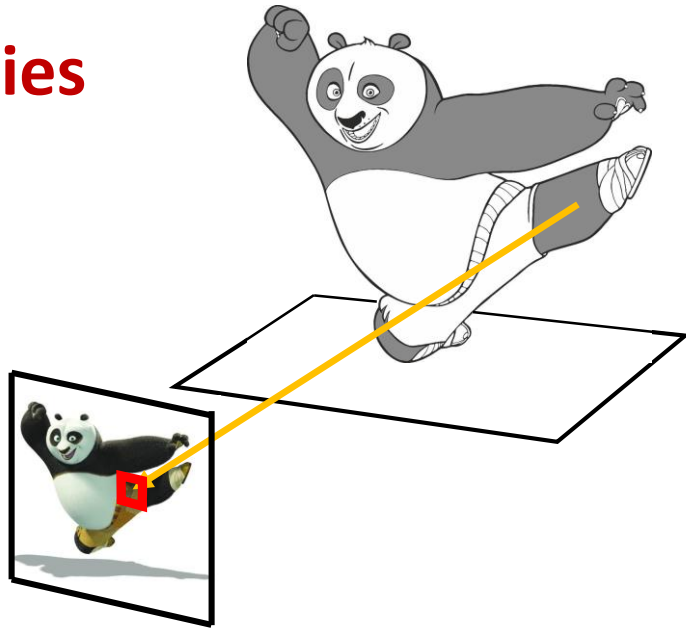


To The Rendering Equation

- What is the **color** of pixel (i,j)?

II

intensities of light at certain frequencies



Balance Equation

- Accountability
  - [*outgoing*] - [*incoming*] = [*emitted*] - [*absorbed*]
- Macro level
  - *The total light energy put into the system must equal the energy leaving the system (usually, via heat).*

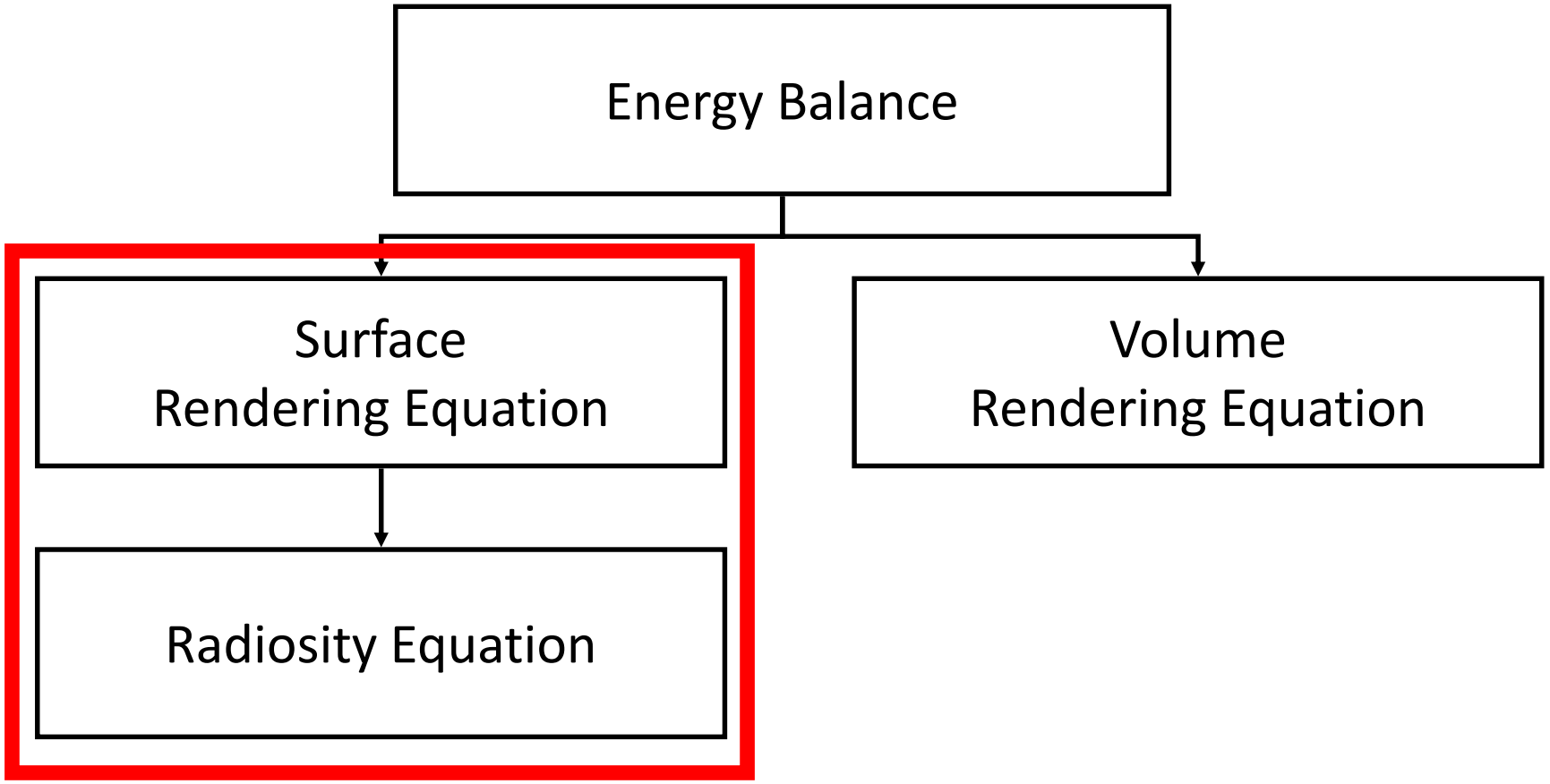
$$\Phi_o - \Phi_i = \Phi_e - \Phi_a$$

- Micro level
  - *The energy flowing into a small region of phase space must equal the energy flowing out.*

$$B(x) - E(x) = B_e(x) - E_a(x)$$

$$L_o(x,\omega) - L_i(x,\omega) = L_e(x,\omega) - L_a(x,\omega)$$

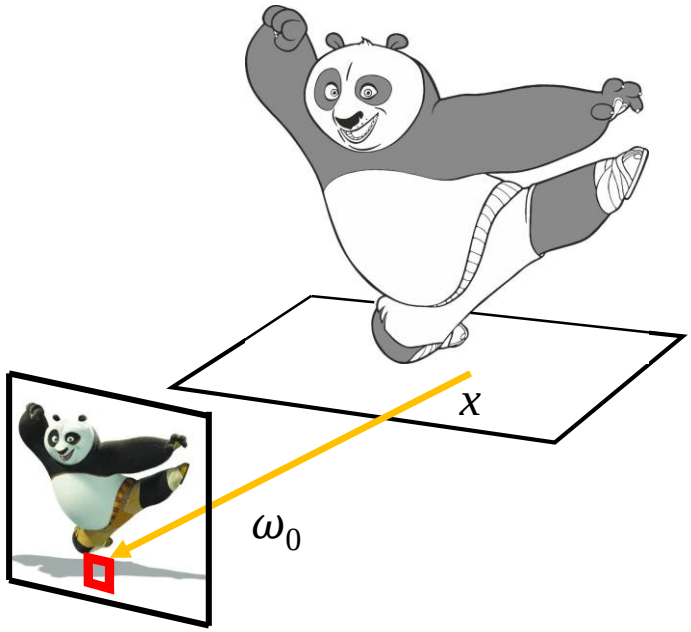
Rendering Equations



Surface Rendering Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

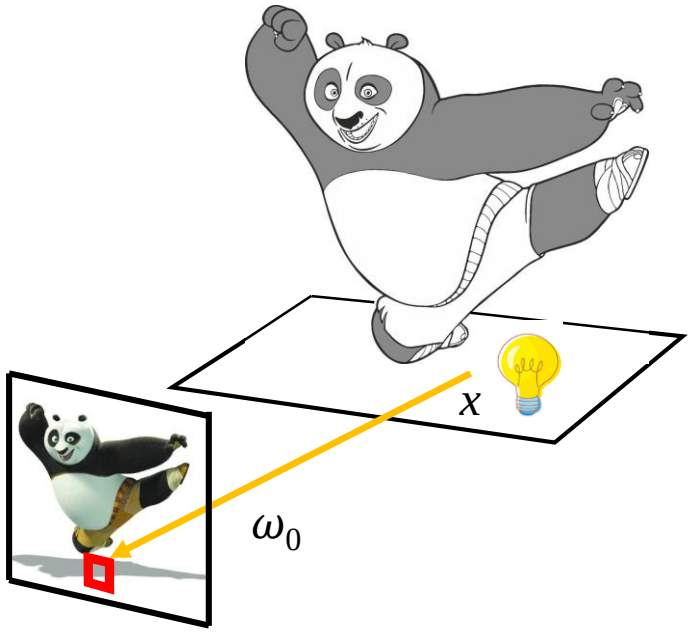


Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$

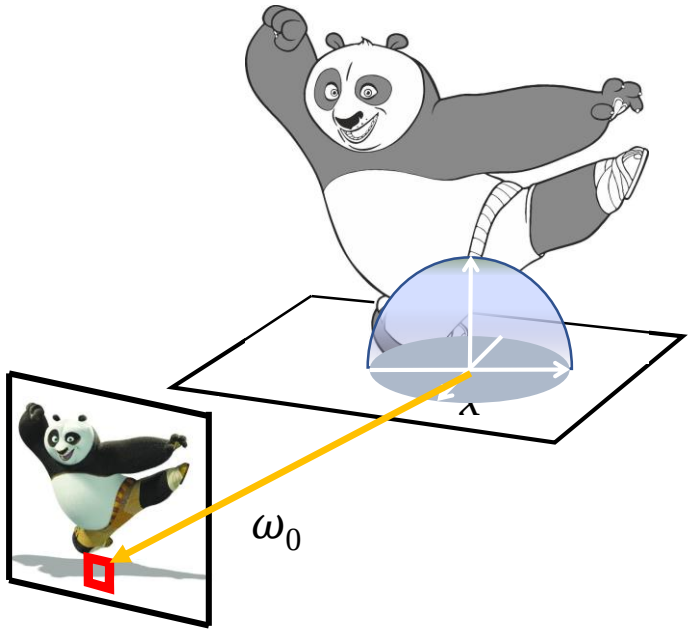
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

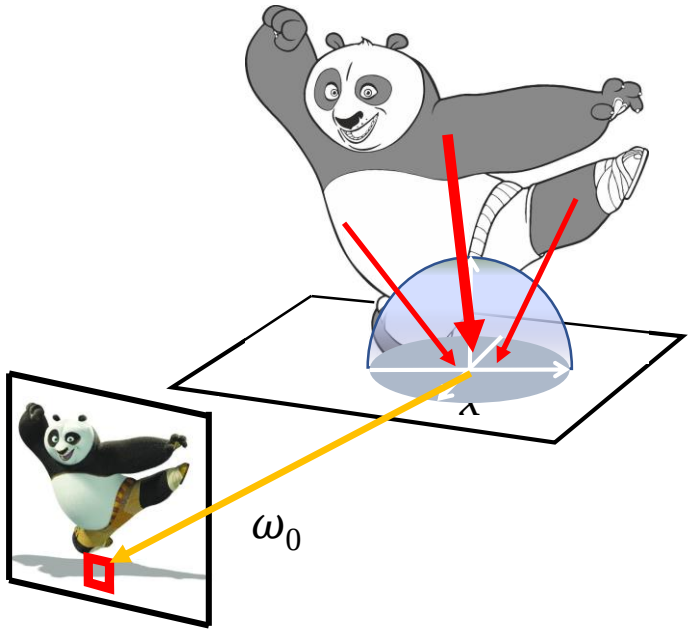
$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

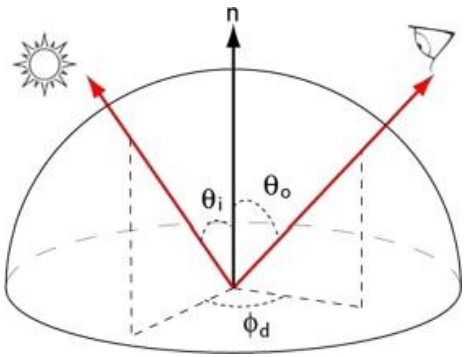
$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \to \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

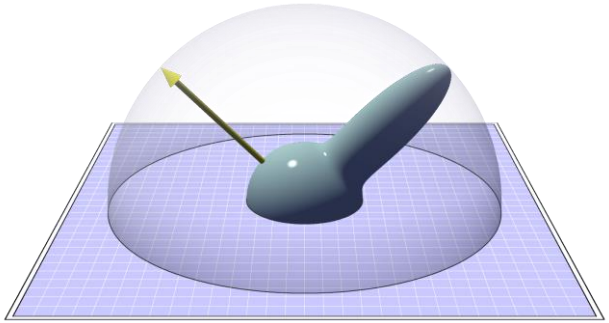


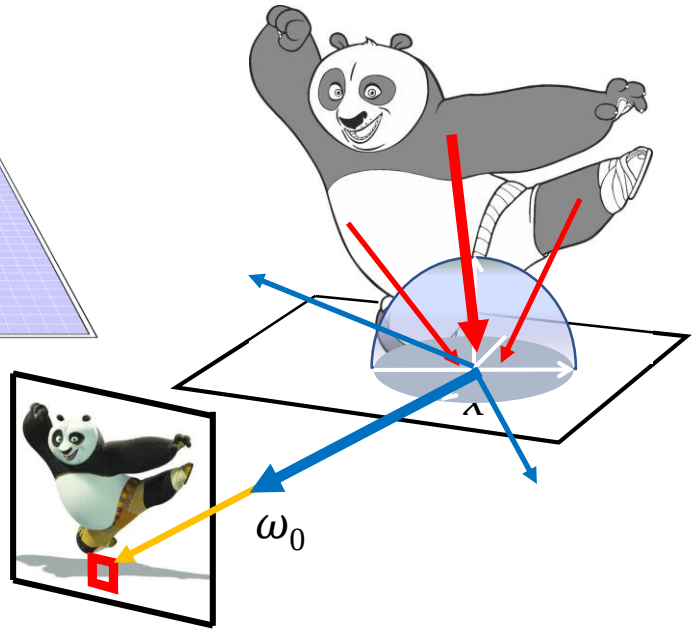
Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



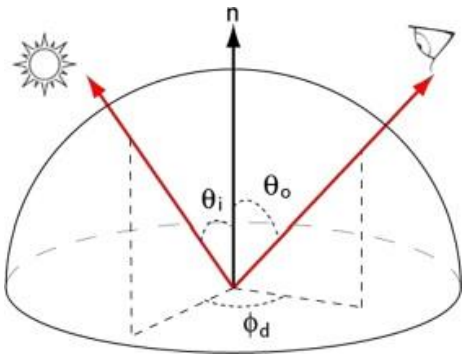


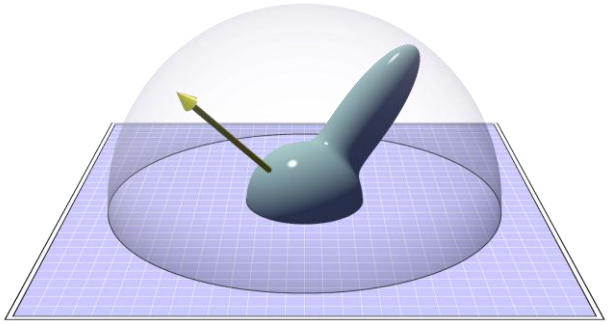


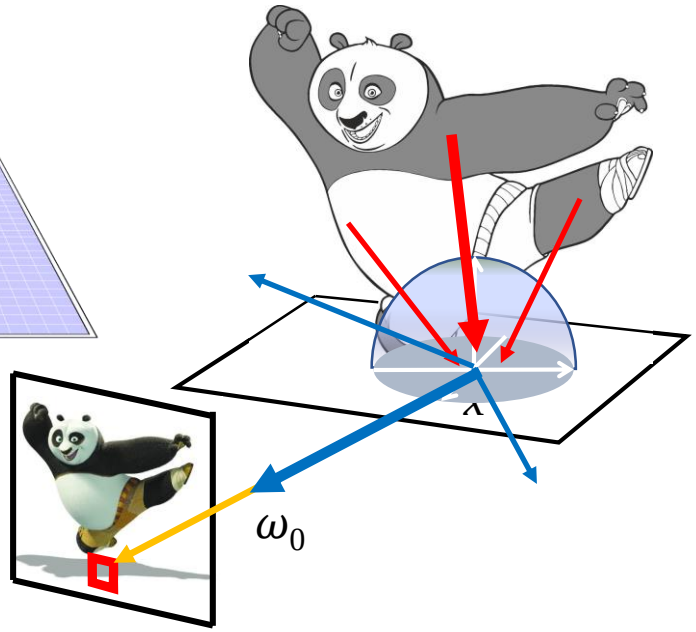
Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



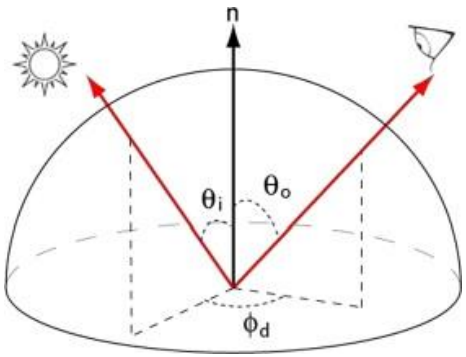


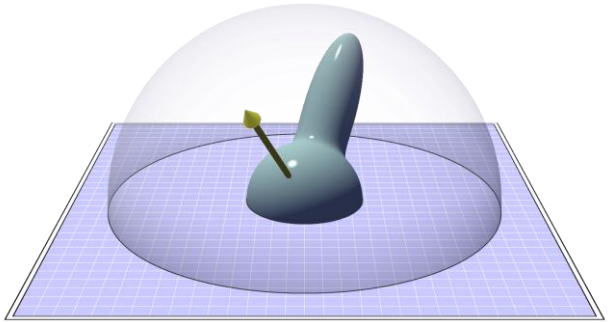


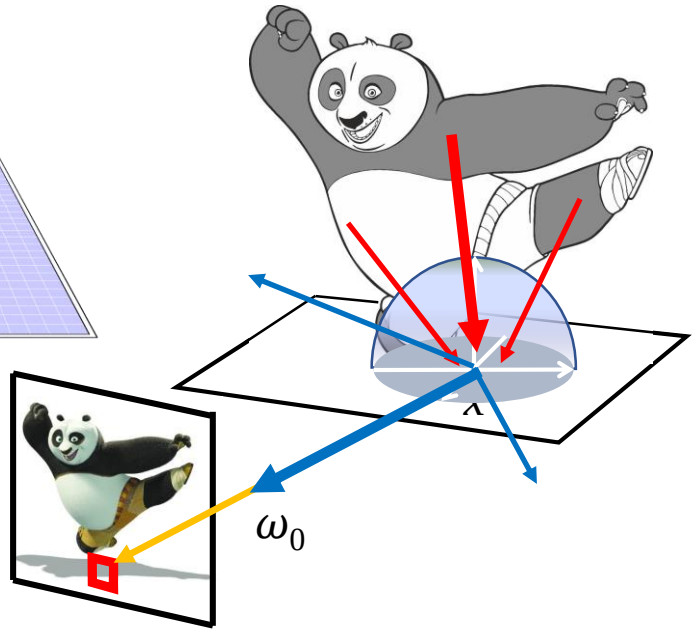
Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



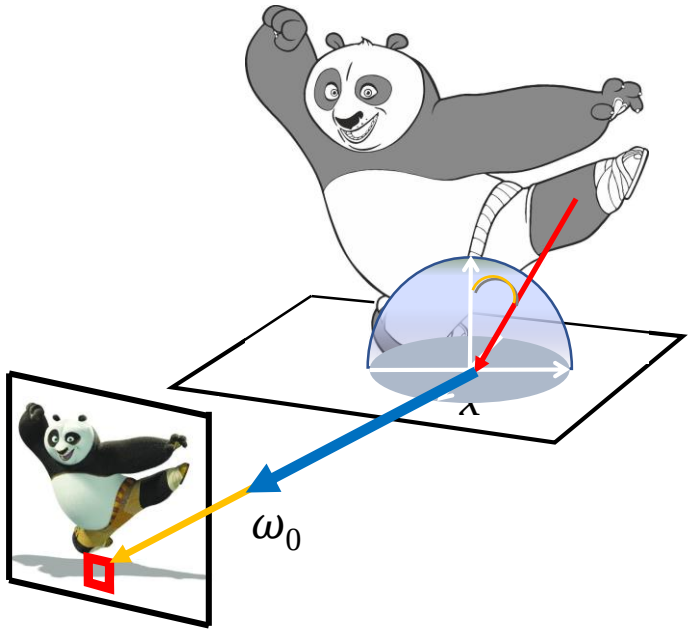




Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

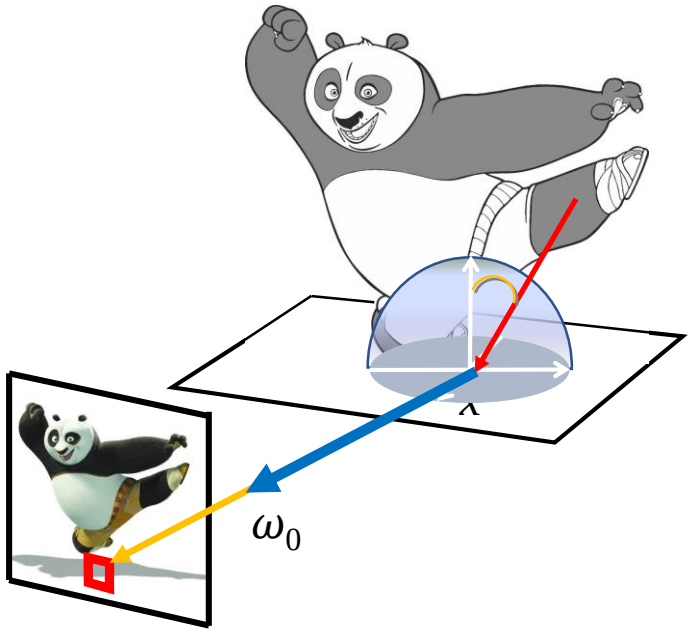
$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



Surface Balance Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

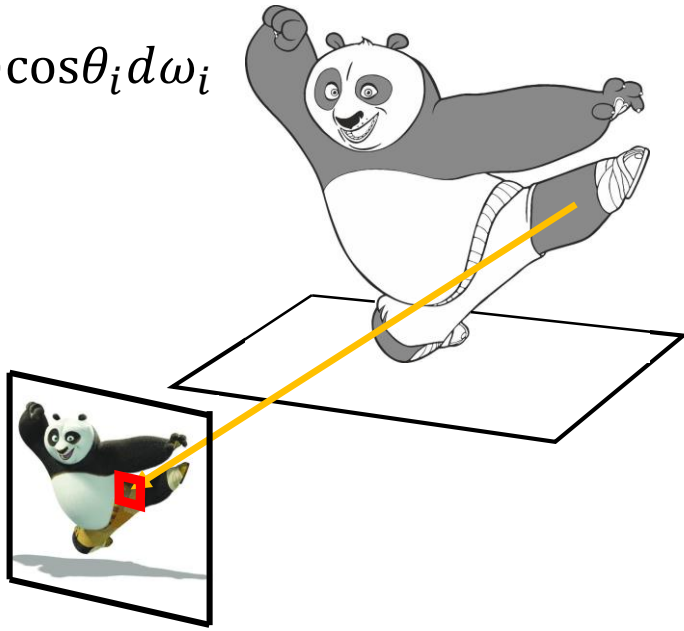


Solving Rendering Equation

Solving Rendering Equation

- What is the color of pixel (i,j)?

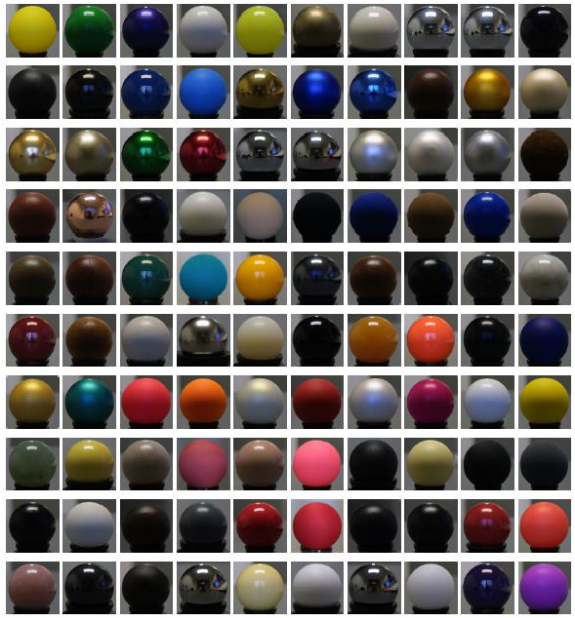
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

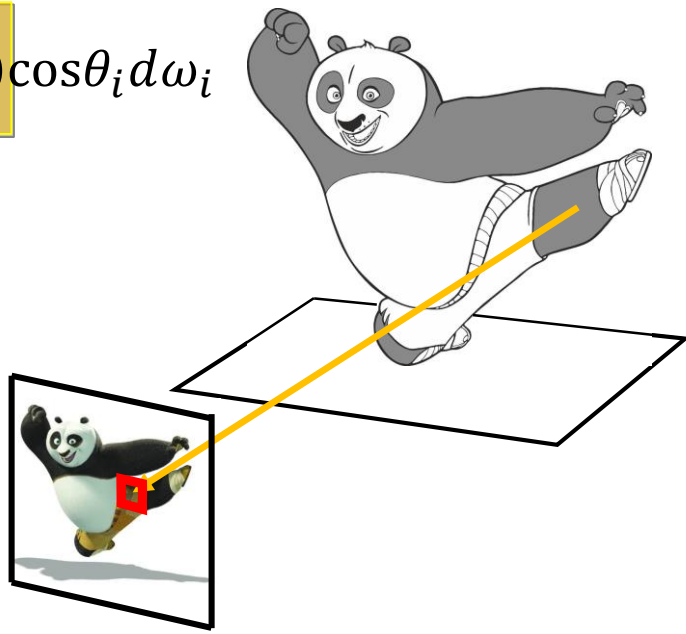


Solving Rendering Equation

- What is the color of pixel (i,j)?

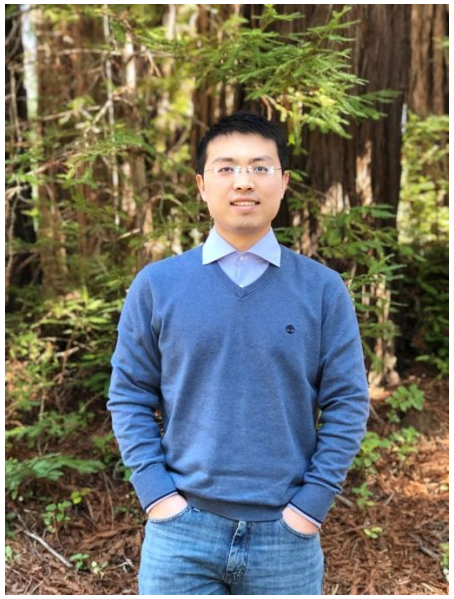
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$





Rendering Tutorial III: - Materials

闫令琪



Abstract

照片级的真实感图形渲染正变得越来越重要。不论是在电影还是游戏行业中，图形渲染的质量都能够直接决定整个产品的成败。然而，现今的图形渲染生成的图片仍然因为看上去过于完美而显得非常不真实。在本报告中，我会介绍一种基于微观细节的表面建模和渲染方法，用实际的微观表面取代以往在计算机图形学中被广泛使用的统计学意义上的物体表面，从而极大地提升真实感。具体而言，我会讨论微观细节的表示，复杂表面反射模型的建立和加速渲染，以及几何光学和波动光学的不同建模方法。

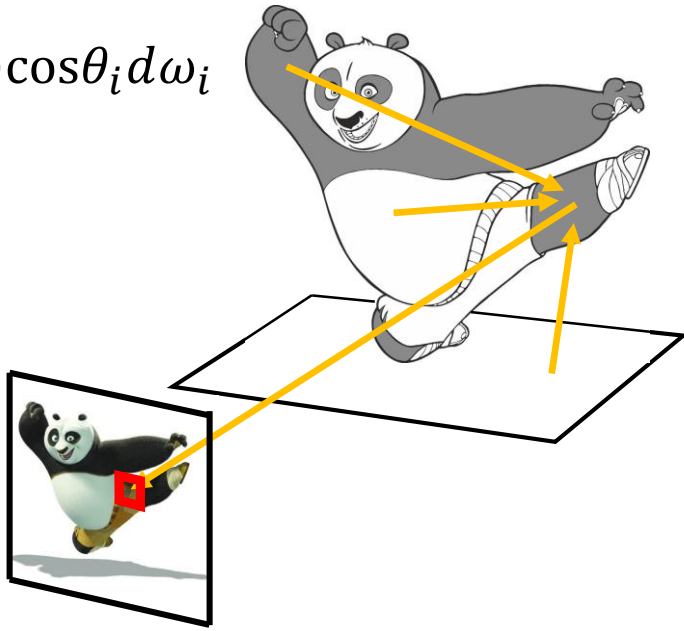
Bio

加州大学圣芭芭拉分校助理教授。闫令琪博士于 2018 年获加州大学伯克利分校博士学位，2013年获清华大学学士学位。他的主要研究方向是基于物理的真实感图形渲染及其相关的数学和物理理论，具体包括基于微观细节的表面观测和建模、离线和实时光线追踪、信号的采样和重建、高效的光线传播和散射等等。闫令琪博士开创并启发了一系列下一代计算机图形学的研究方向，如高度细致的渲染和实时光线追踪，并于2018年因其开创性研究被授予C.V. Ramamoorthy 杰出科研奖。此外，他的科研成果还被直接应用于电影和游戏业，曾帮助影片

Solving Rendering Equation

- What is the color of pixel (i,j)?

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

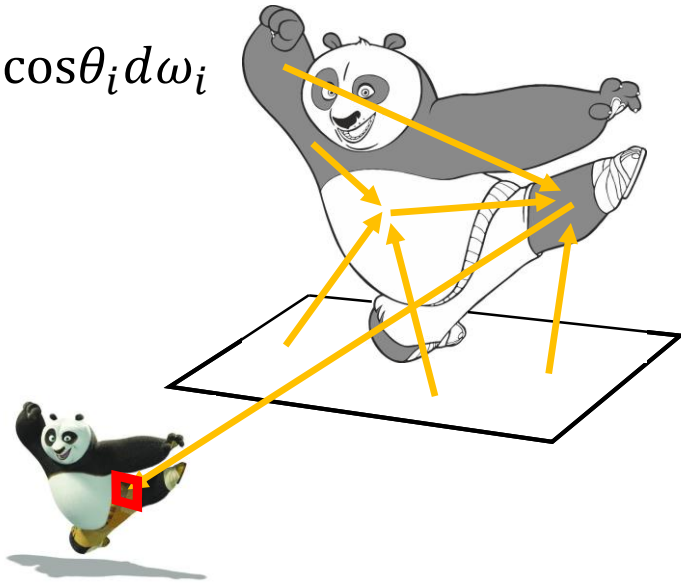


Solving Rendering Equation

- What is the color of pixel (i,j)?

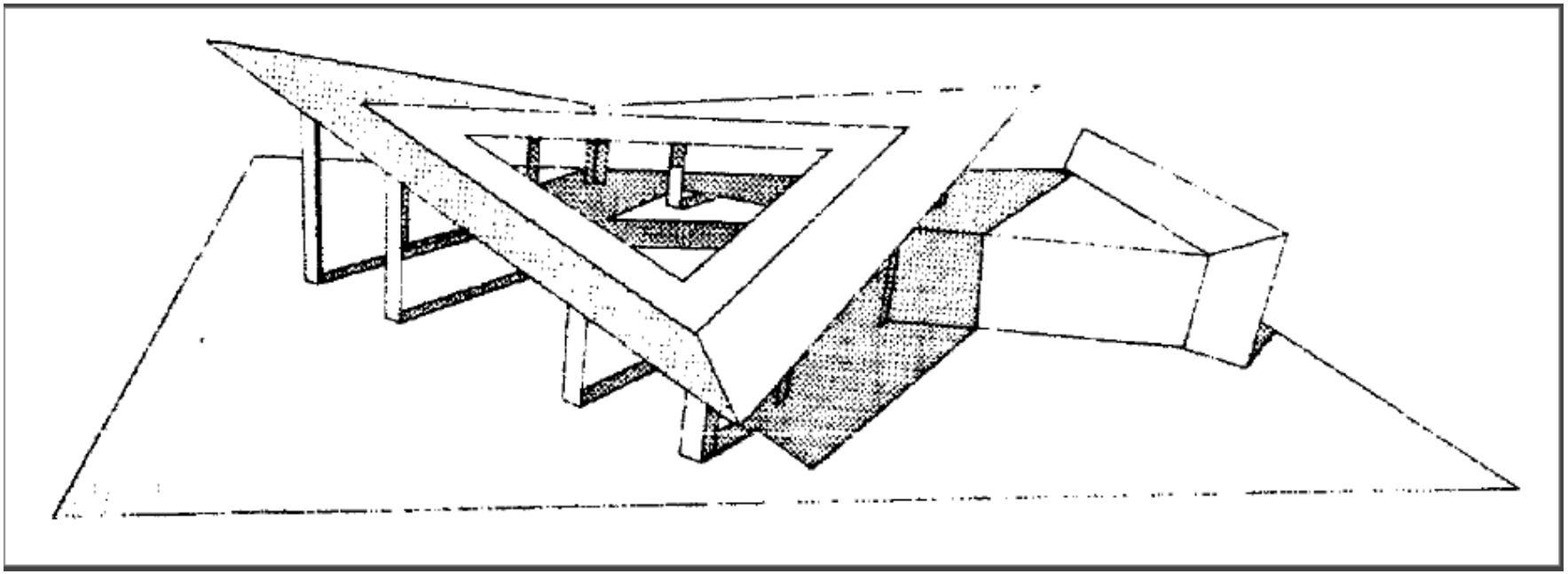
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$



Solving Rendering Equation

- Ray tracing for shadows



Appel 1968

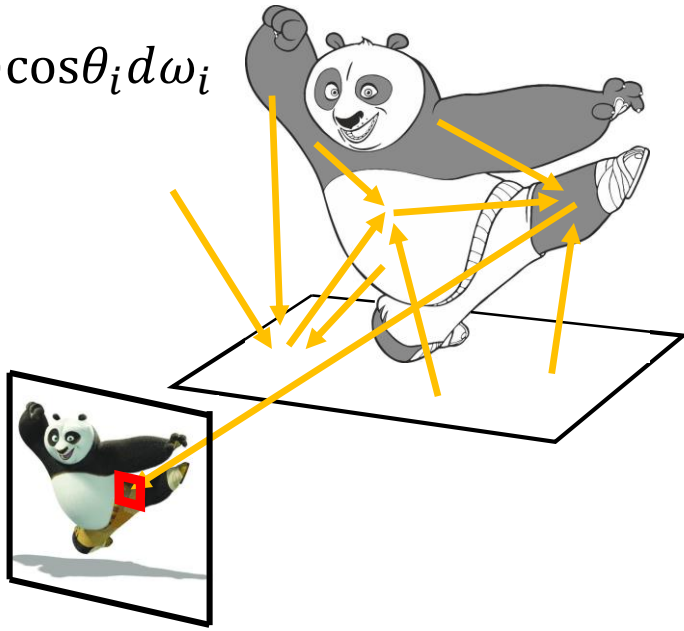
Solving Rendering Equation

- What is the color of pixel (i,j)?

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$



Solving Rendering Equation

- What is the color of pixel (i,j)?

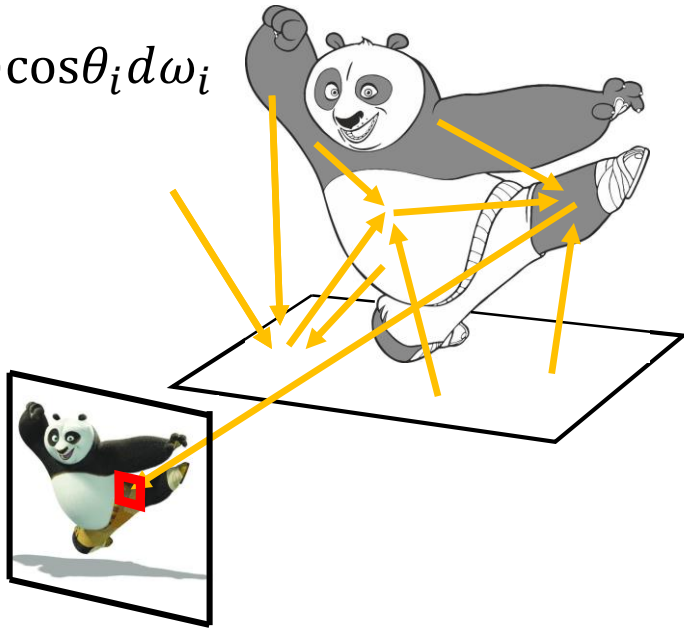
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$

$$\vdots$$

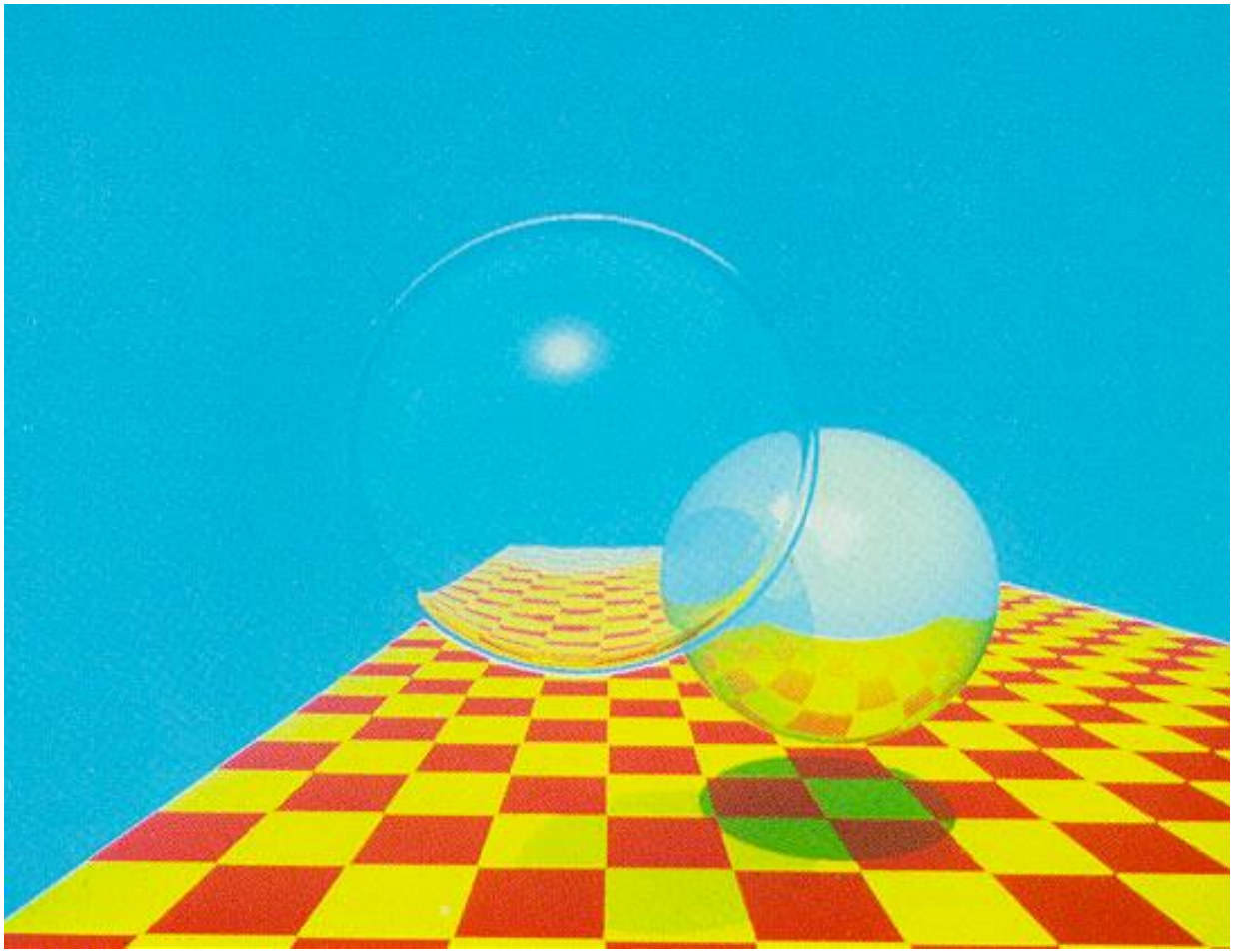
$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o)\cos\theta_i d\omega_i$$



Solving Rendering Equation

- Recursive ray tracing

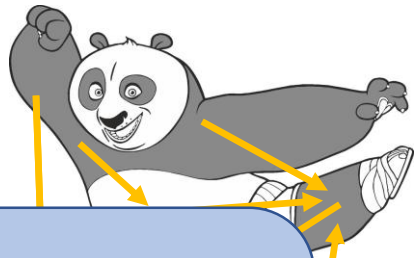
Whitted 1980



Solving Rendering Equation

- What is the color of pixel (i,j)?

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \to \omega_o) \cos\theta_i d\omega_i$$



Curse of dimensionality!!!

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \to \omega_o) \cos\theta_i d\omega_i$$

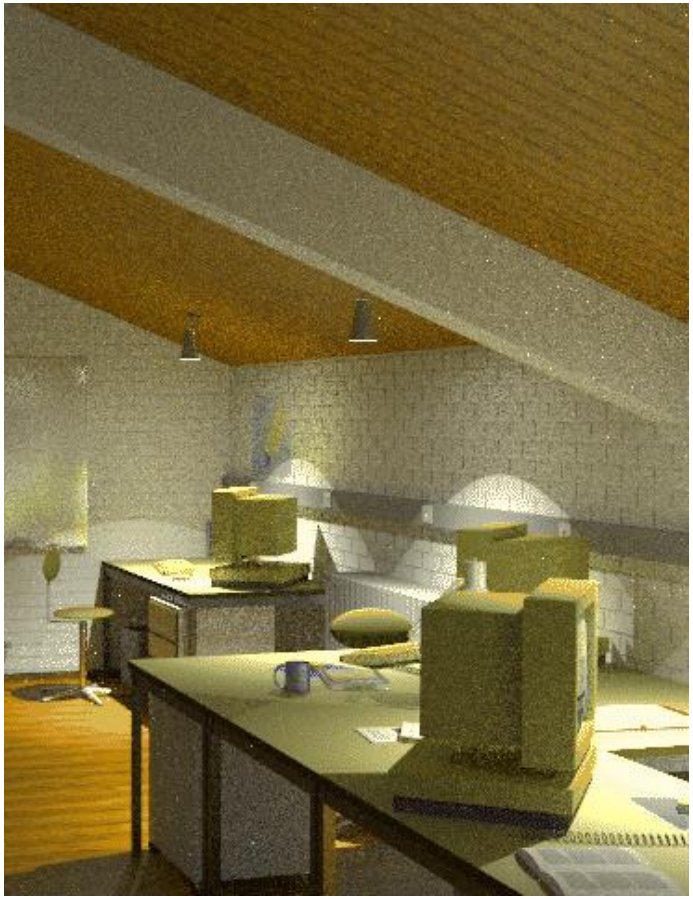
Solving Rendering Equation

- Stochastic ray tracing

Cook, Porter, Carpenter 1984

Solving Rendering Equation

- Bidirectional path tracing

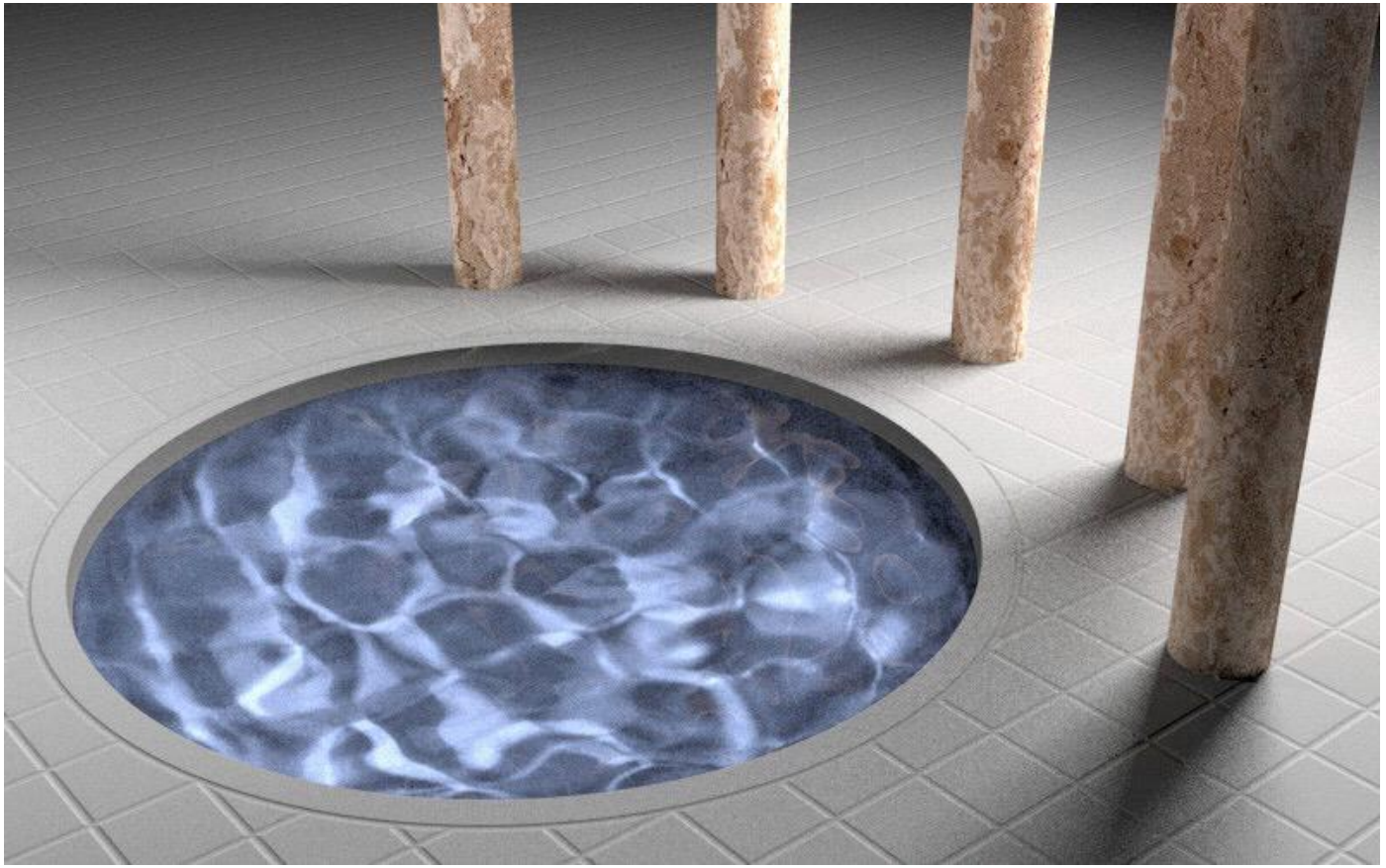


Lafortune and Willems 1993

Veach and Guibas 1994

Solving Rendering Equation

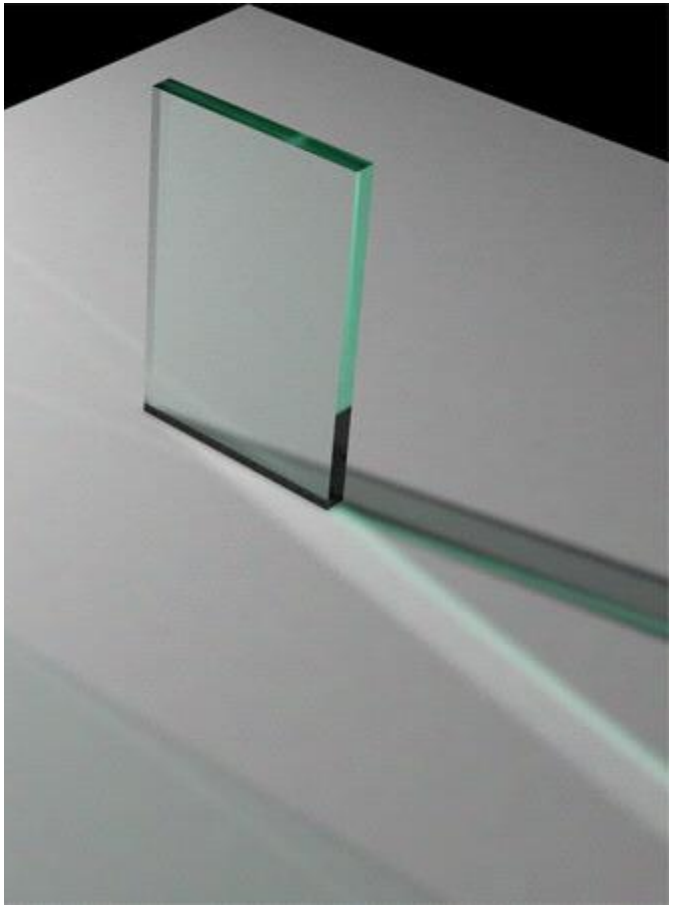
- Metropolis Light Transport (MCMC)



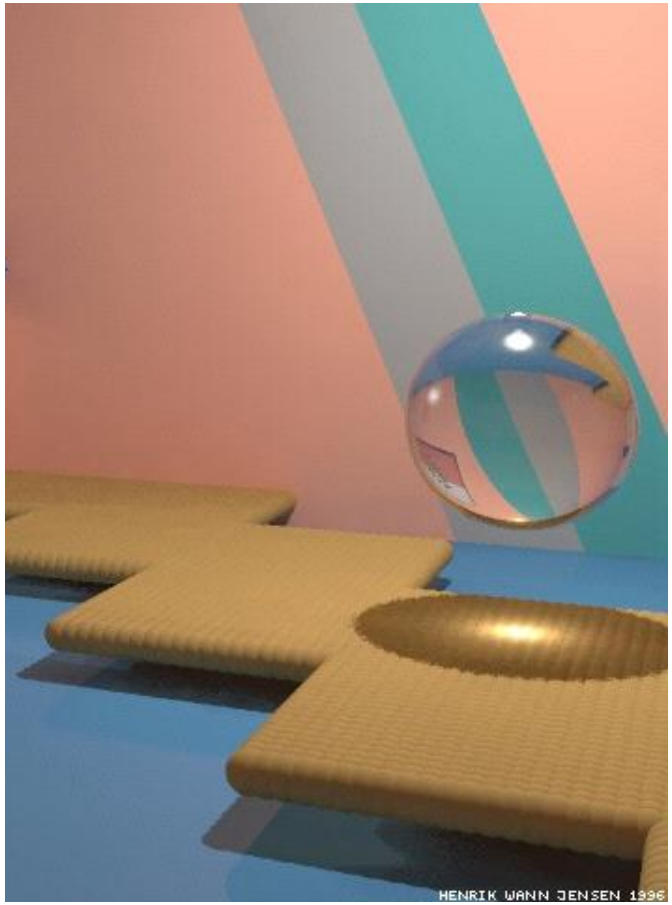
Veach and Guibas 1997

Solving Rendering Equation

- Photon Mapping



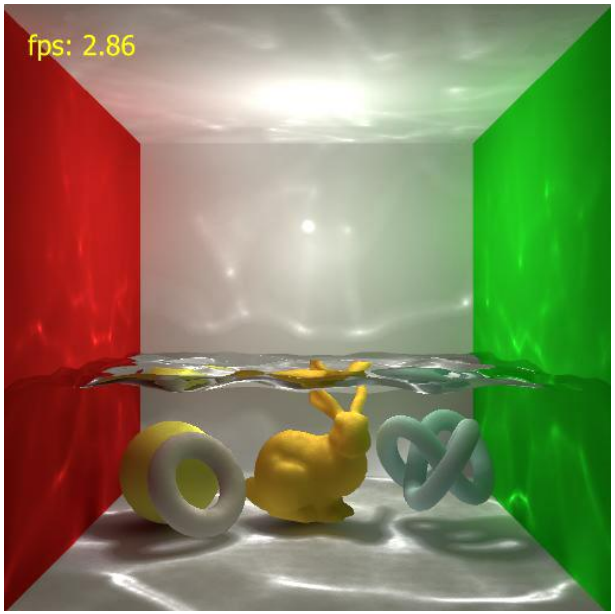
Walter et al. 1997

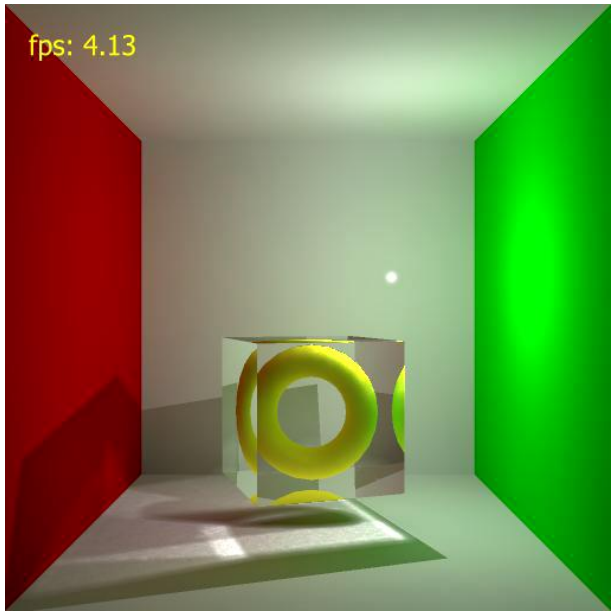


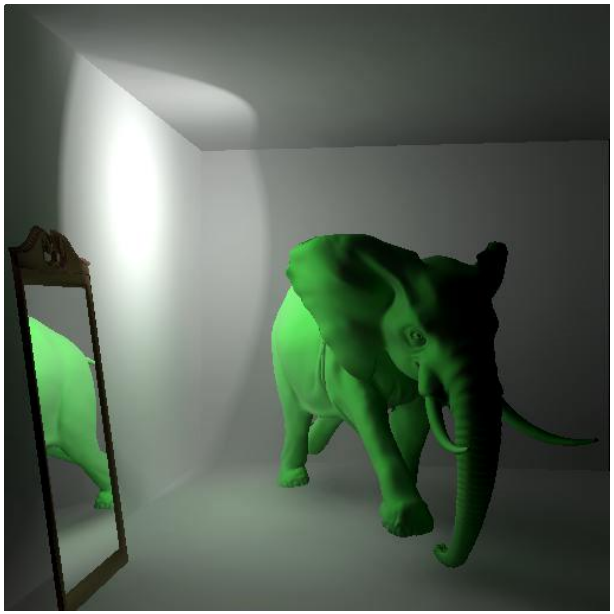
Jensen 1996

Solving Rendering Equation

- GPU-based Photon Mapping







Wang et al. 2009

Solving Rendering Equation

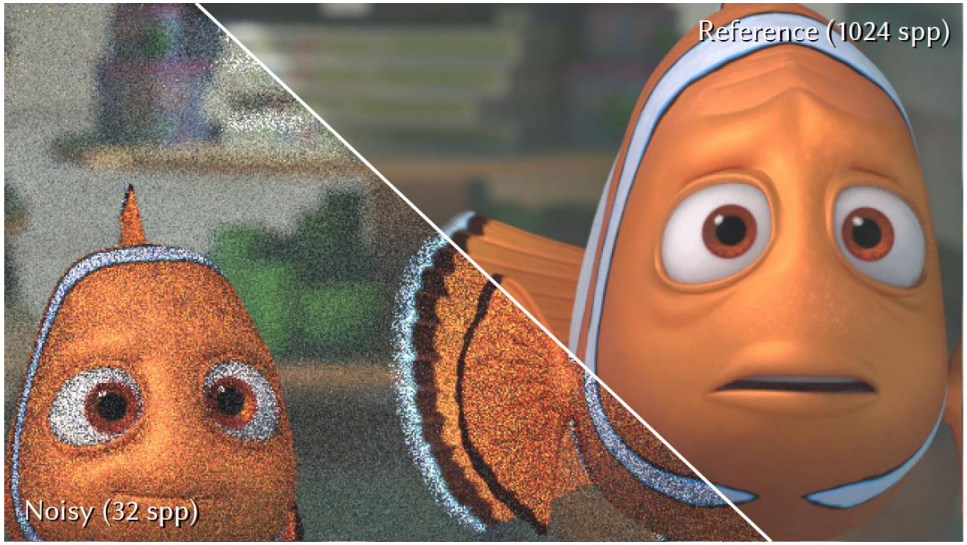
- Vertex Merging and Connection / Unified Path Sampling

Georgiev et al. 2012

Hachisuka 2012

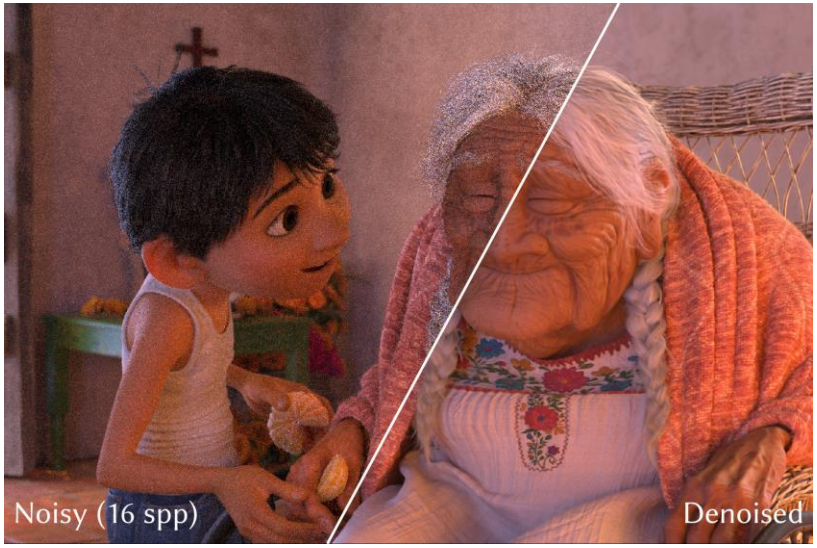
Solving Rendering Equation

- Denoiser using NN



TRAINING

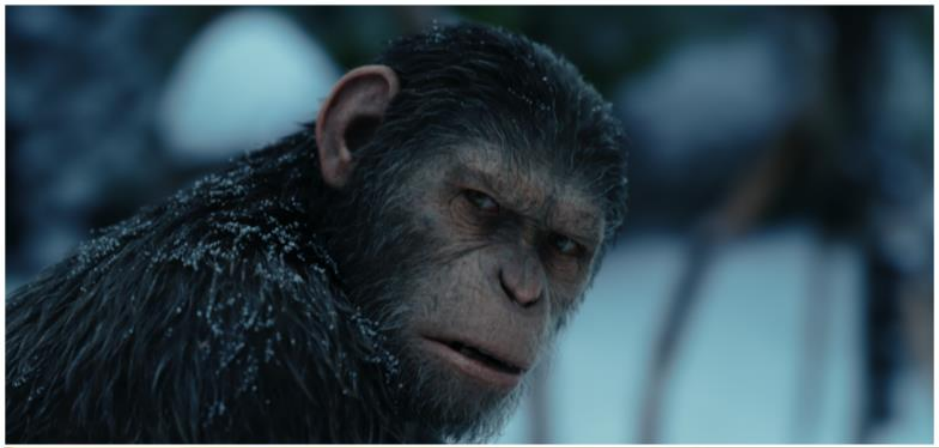
Bako et al. 2017



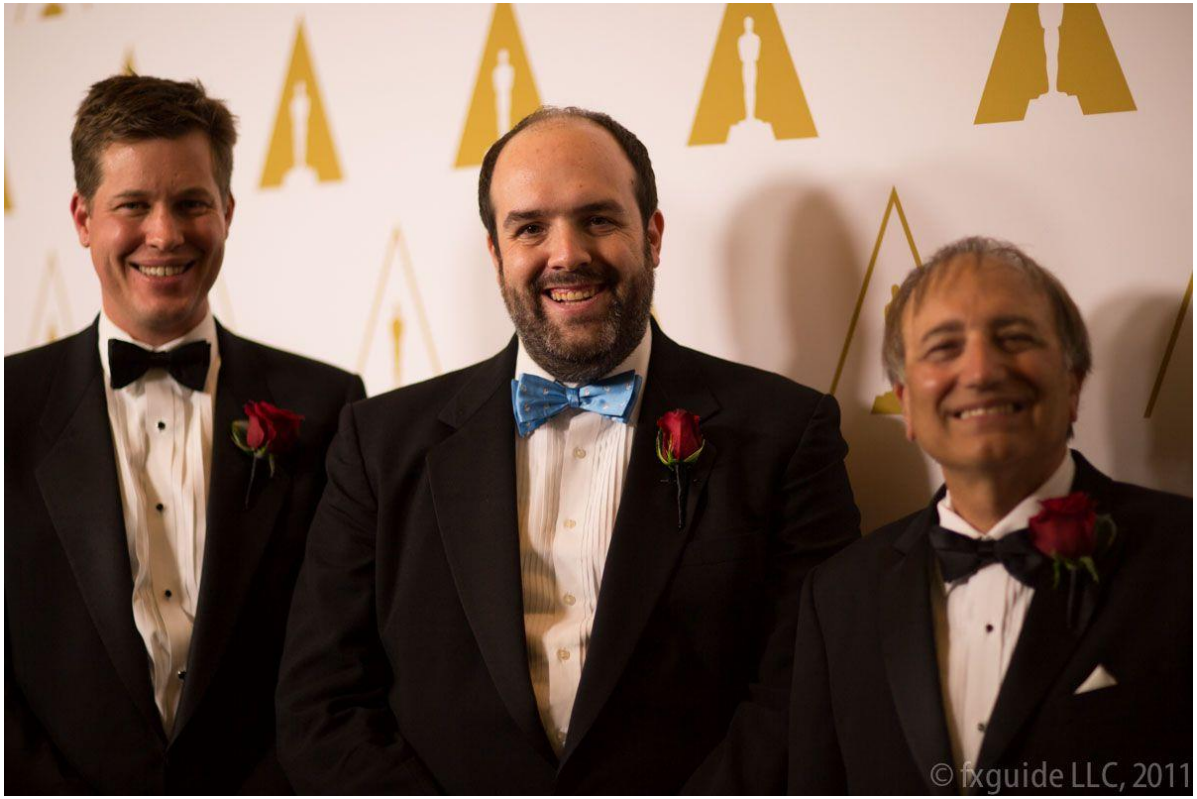
Vogels 2018

Solving Rendering Equation

- Path tracing in Production




Solving Rendering Equation



PBRT

Eric Veach for his research on Monte Carlo path tracing

Oscars 2014 Scientific and Technical Achievement Awards

Rendering Tutorial II: 赵爽 - Monte Carlo Path Tracing

Abstract:

Monte Carlo methods have been the "gold standard" for solving light transport problems due to their ability to offer (1) unbiased estimations and (2) quantitative error analyses. In this lecture, we first introduce Monte Carlo integration as a general framework and then show how it can be applied to solve light transport and rendering problems.

Bio:

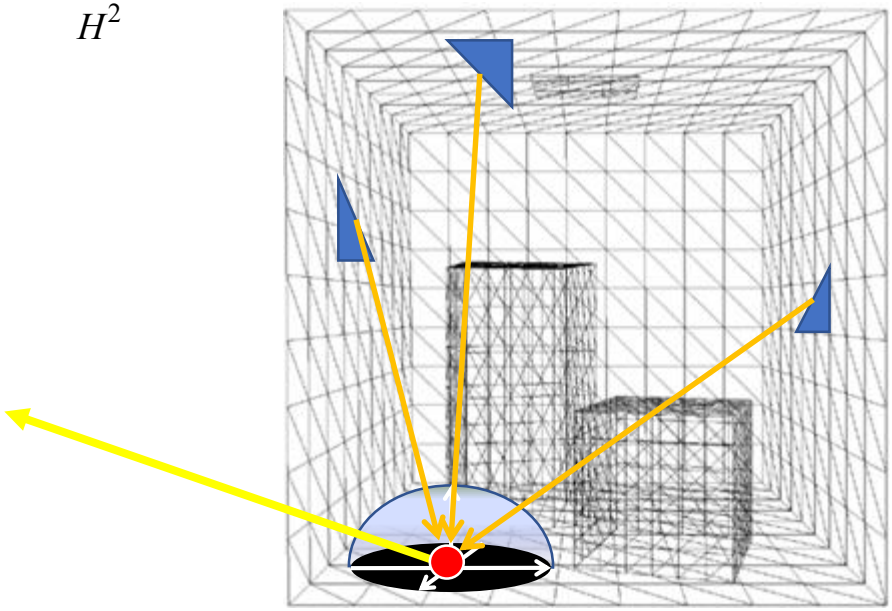
Shuang Zhao is an assistant professor at University of California, Irvine (UCI).

Shuang works in computer graphics, (low-level) vision, and applied perception. He is interested in modeling and solving particle (e.g., photon and neutron) transport problems under complex and realistic configurations. By drawing inspirations from optics, material science, and computational statistics, his group has been developing new techniques for computer graphics & vision, industrial design, material engineering, and computational biophotonics applications.

Another Form of Rendering Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

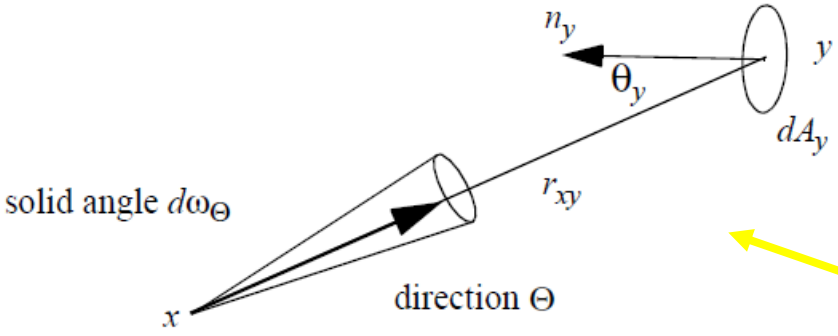
$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \rightarrow \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

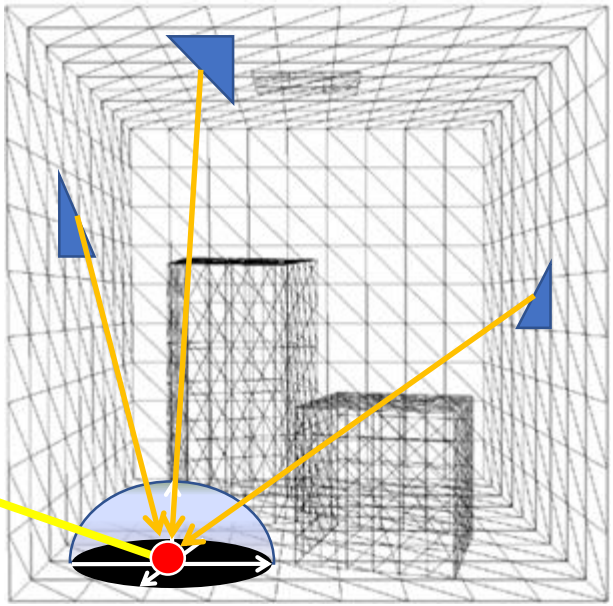


Another Form of Rendering Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L_o(x,\omega_o) = L_e(x,\omega_o) + L_r(x,\omega_o)$$
$$= L_e(x,\omega_o) + \int_{H^2} f_r(x,\omega_i \to \omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$



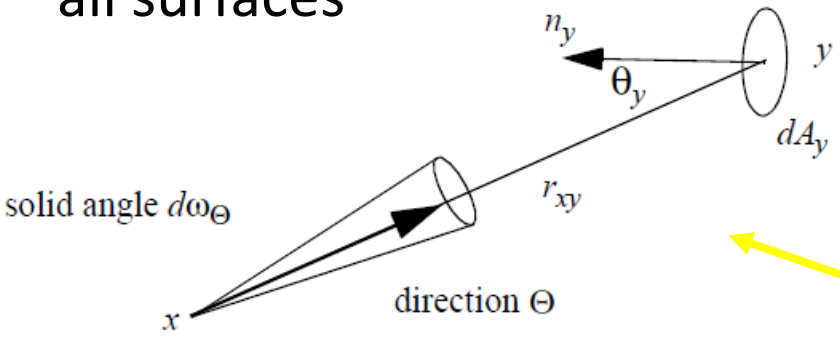


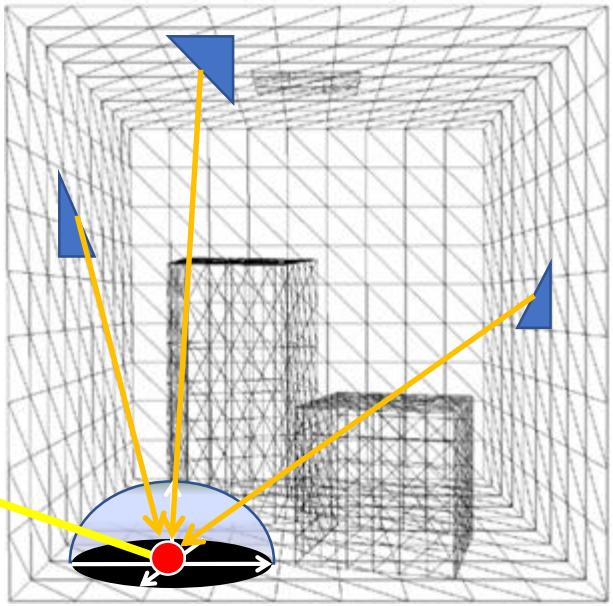
Another Form of Rendering Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L(x', x) = L_e(x', x) + \int_{M^2} f_r(x'', x', x)\, L(x'', x')\, G(x'', x')\, dA''(x'')$$

Integrate over all surfaces





Another Form of Rendering Equation

- [*outgoing*] = [*emitted*] + [*reflected*]

$$L(x', x) = L_e(x', x) + \int_{M^2} f_r(x'', x', x)\, L(x'', x')\, G(x'', x')\, dA''(x'')$$

Integrate over all surfaces

Geometry term

$$G(x'', x') = \frac{\cos\theta_i'' \cos\theta_o'}{\|x'' - x'\|^2} V(x'', x')$$

Visibility term

$$V(x'', x') = \begin{cases} 1 & \text{visible} \\ 0 & \text{not visible} \end{cases}$$

Solving the Rendering Equation

- Rendering Equations

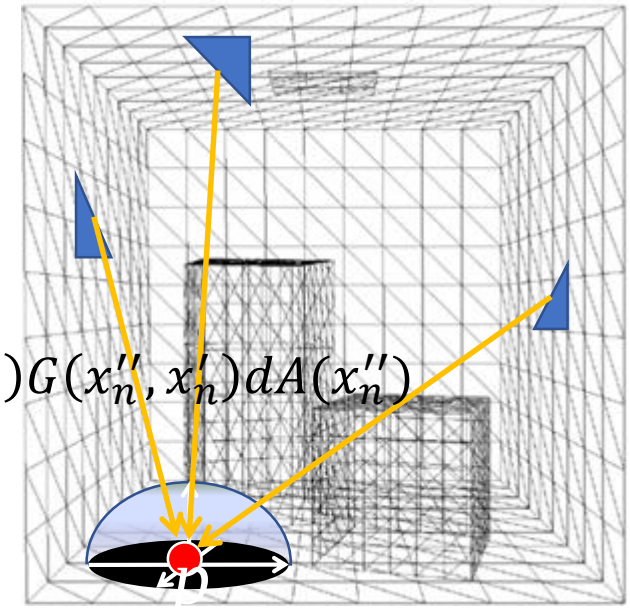
$$L_o(x_0', x_0) = L_e(x_0', x_0) + \int_{M^2} L_i(x_0', x_0'') f_r(x_0, x_0', x_0'') G(x_0'', x_0') dA(x_0'')$$

$$L_o(x_1', x_1) = L_e(x_1', x_1) + \int_{M^2} L_i(x_1', x_1'') f_r(x_1, x_1', x_1'') G(x_1'', x_1') dA(x_1'')$$

$$\vdots$$

$$\vdots$$

$$L_o(x_n', x_n) = L_e(x_n', x_n) + \int_{M^2} L_i(x_n', x_n'') f_r(x_n, x_n', x_n'') G(x_n'', x_n') dA(x_n'')$$



Solving the Rendering Equation

- Rendering Equations

$$K \equiv S \circ T$$

$$L_o(x_0', x_0) = L_e(x_0', x_0) + \int_{M^2} L_i(x_0', x_0'') f_r(x_0, x_0', x_0'') G(x_0'', x_0') dA(x_0'')$$

$$L_o(x_1', x_1) = L_e(x_1', x_1) + \int_{M^2} L_i(x_1', x_1'') f_r(x_1, x_1', x_1'') G(x_1'', x_1') dA(x_1'')$$

$$\vdots$$

$$L = L_e + K \circ L$$

$$\vdots$$

$$L_o(x_n', x_n) = L_e(x_n', x_n) + \int_{M^2} L_i(x_n', x_n'') f_r(x_n, x_n', x_n'') G(x_n'', x_n') dA(x_n'')$$

Solving the Rendering Equation

- Rendering Equations

$$K \equiv S \circ T$$

$$L_o(x_0', x_0) = L_e(x_0', x_0) + \int_{M^2} L_i(x_0', x_0'') f_r(x_0, x_0', x_0'') G(x_0'', x_0') dA(x_0'')$$

$$L_o(x_1', x_1) = L_e(x_1', x_1) + \int_{M^2} L_i(x_1', x_1'') f_r(x_1, x_1', x_1'') G(x_1'', x_1') dA(x_1'')$$

$$\vdots$$

$$L = L_e + K \circ L \Rightarrow (I - K) \circ L = L_e$$

$$\vdots$$

$$L_o(x_n', x_n) = L_e(x_n', x_n) + \int_{M^2} L_i(x_n', x_n'') f_r(x_n, x_n', x_n'') G(x_n'', x_n') dA(x_n'')$$

Solving the Rendering Equation

- Rendering Equations $K \equiv S \circ T$

$$L_o(x_0', x_0) = L_e(x_0', x_0) + \int_{M^2} L_i(x_0', x_0'') f_r(x_0, x_0', x_0'') G(x_0'', x_0') dA(x_0'')$$

$$L_o(x_1', x_1) = L_e(x_1', x_1) + \int_{M^2} L_i(x_1', x_1'') f_r(x_1, x_1', x_1'') G(x_1'', x_1') dA(x_1'')$$

$$L = L_e + K \circ L \Rightarrow (I - K) \circ L = L_e$$

- Solution

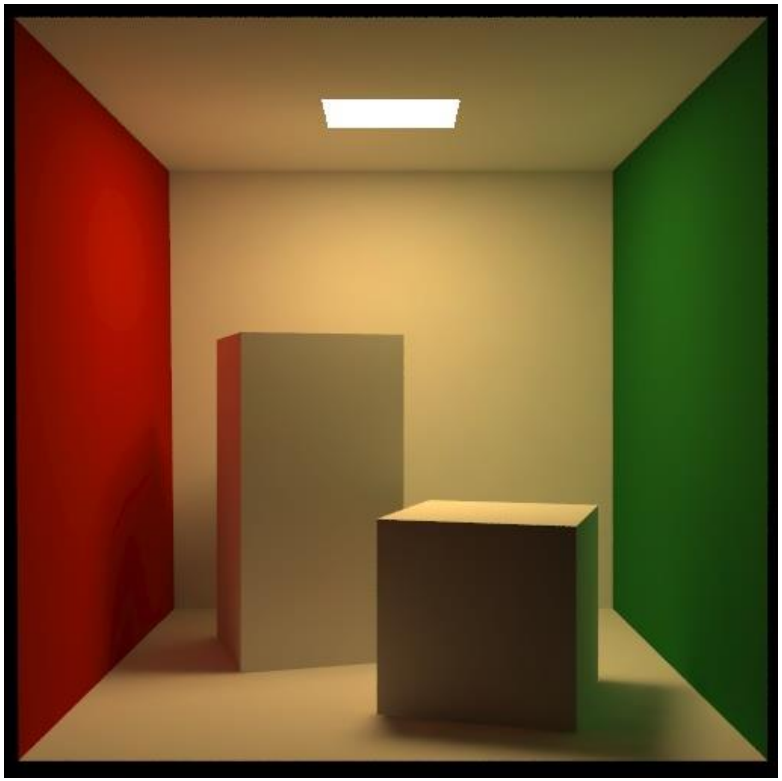
$$L_o(x_n', x_n) = L_e(x_n', x_n) + \int_{M^2} L_i(x_n', x_n'') f_r(x_n, x_n', x_n'') G(x_n'', x_n') dA(x_n'')$$

$$L = (I - K)^{-1} \circ L_e$$

Solving the Rendering Equation

- Radiosity

$$L = (I - K)^{-1} \circ L_e$$



Goral et al. 1984

$$B_j = E_j + \rho_j \sum_{i=1}^{N} B_i F_{ij}\,, \qquad j = 1 \ldots N$$

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \cdots & 1-\rho_N F_{NN} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{bmatrix}$$

$$\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix} = \begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \cdots & 1-\rho_N F_{NN} \end{bmatrix}^{-1} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{bmatrix}$$

General / Formal Solution

- However, it is impossible to directly compute $(I - K)^{-1}$ generally.

- Neumann series

$$(I - K)^{-1} = \frac{1}{I - K} = I + K + K^2 + \cdots$$

- Verify

$$\begin{aligned}(I-K)\circ(I-K)^{-1} &= (I-K)\circ(I+K+K^2+\ldots)\\ &= (I+K+\ldots)-(K+K^2+\ldots)\\ &= I\end{aligned}$$

General / Formal Solution

- Successive approximations

$$L^1 = L_e$$

$$L^2 = L_e + K \circ L^1$$

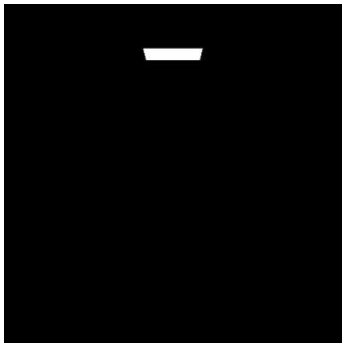
$$\dots$$

$$L^n = L_e + K \circ L^{n-1}$$

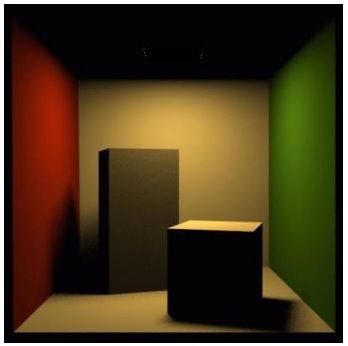
- Converged

$$L^n = L^{n-1} \quad \therefore \quad L^n = L_e + K \circ L^n$$

Successive Approximation



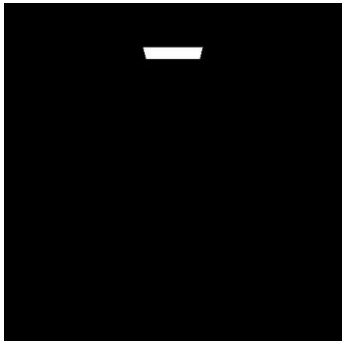
L_e



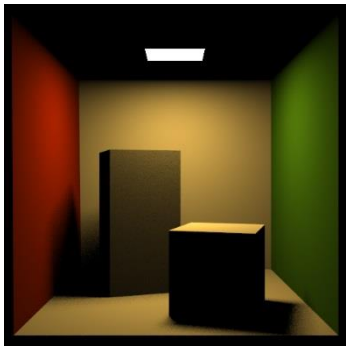
$K \circ L_e$

$K \circ K \circ L_e$

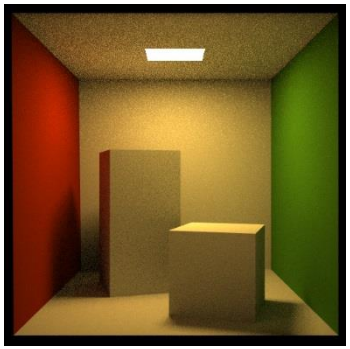
$K \circ K \circ K \circ L_e$



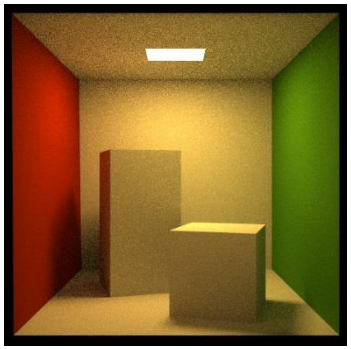
L_e



$L_e + K \circ L_e$



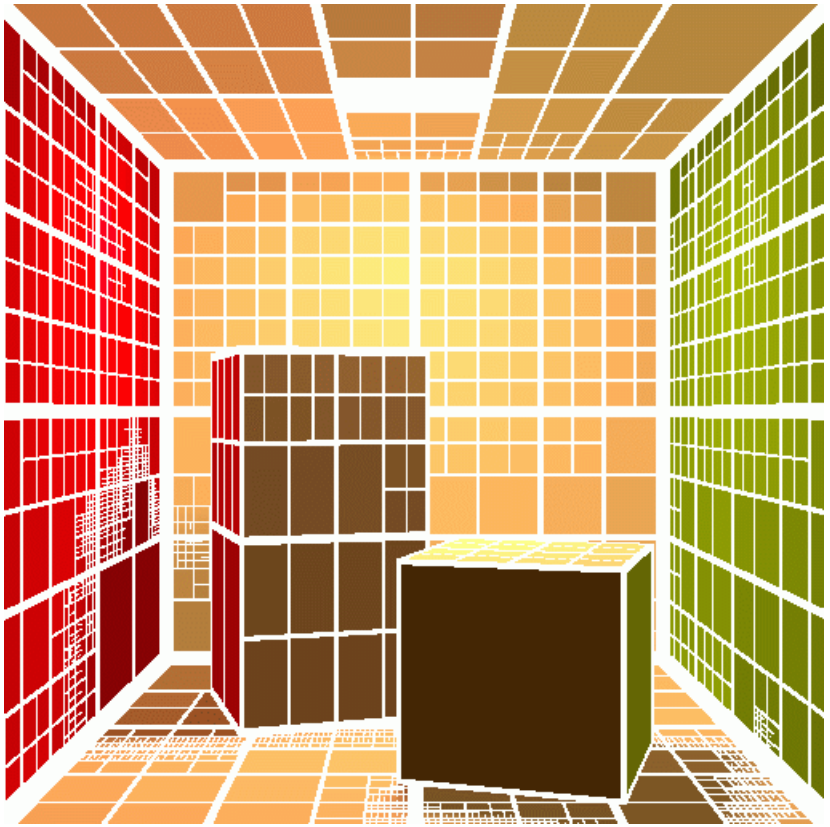
$L_e + \cdots K^2 \circ L_e$



$L_e + \cdots K^3 \circ L_e$

Solving the Rendering Equation

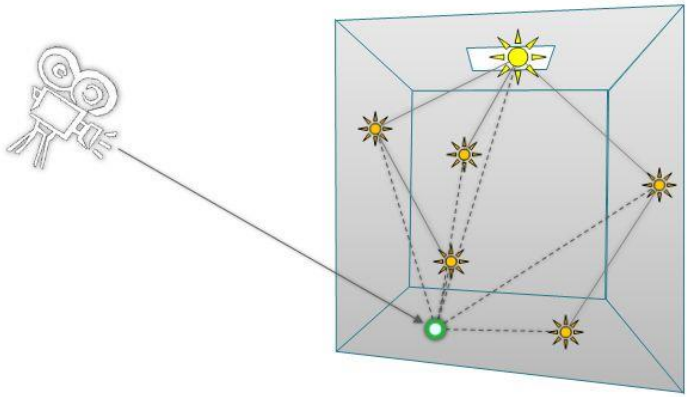
- Hierarchical radiosity



Hanrahan et al. 1991

Solving the Rendering Equation

- Instant Radiosity (Virtual point lights)



Keller 1997

Solving the Rendering Equation

- Many-light rendering (Point GI)

Walter et al. 2005
LightCuts

Solving the Rendering Equation

- Matrix Sampling-and-Recovery via Sparsity Analysis

Huo et al. 2015

Solving the Rendering Equation

- Many-light rendering (Point-based GI)

Pixar

A short recap

- Rendering equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{H^2} L_i(x, \omega_i) f_r(x, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

- Solving the rendering equation as accurate as possible
  - Realistic rendering algorithms: Path tracing, Radiosity…
  - Limitation: very slow

Real-time rendering

- Rendering equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{H^2} L_i(x, \omega_i) f_r(x, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

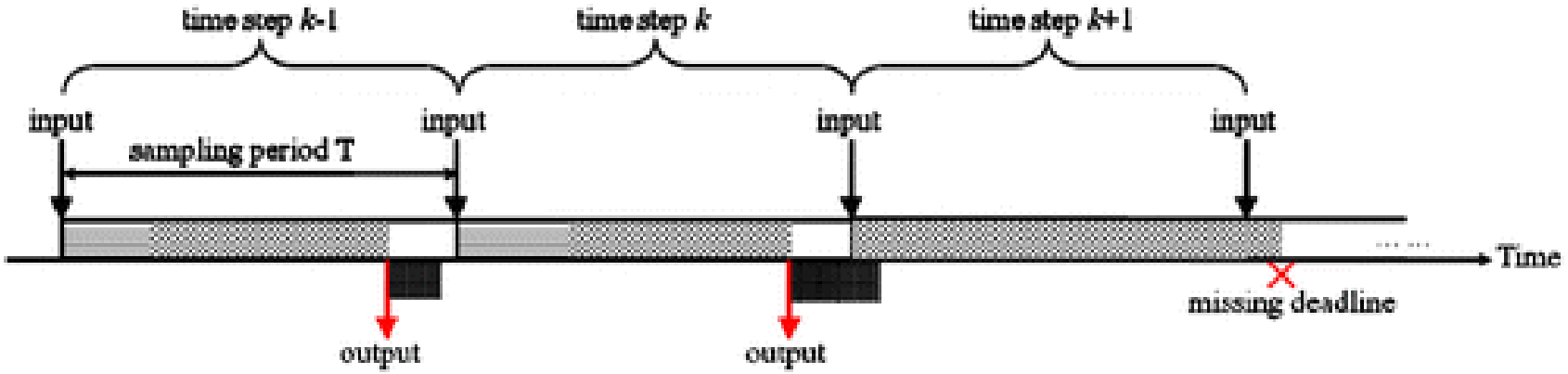
- Solving the rendering equation as accurate as possible
  - Realistic rendering algorithms: Path tracing, Radiosity…
  - Limitation: very slow

- Solving the rendering equation as fast as possible
  - Real-time rendering

Real-time Rendering

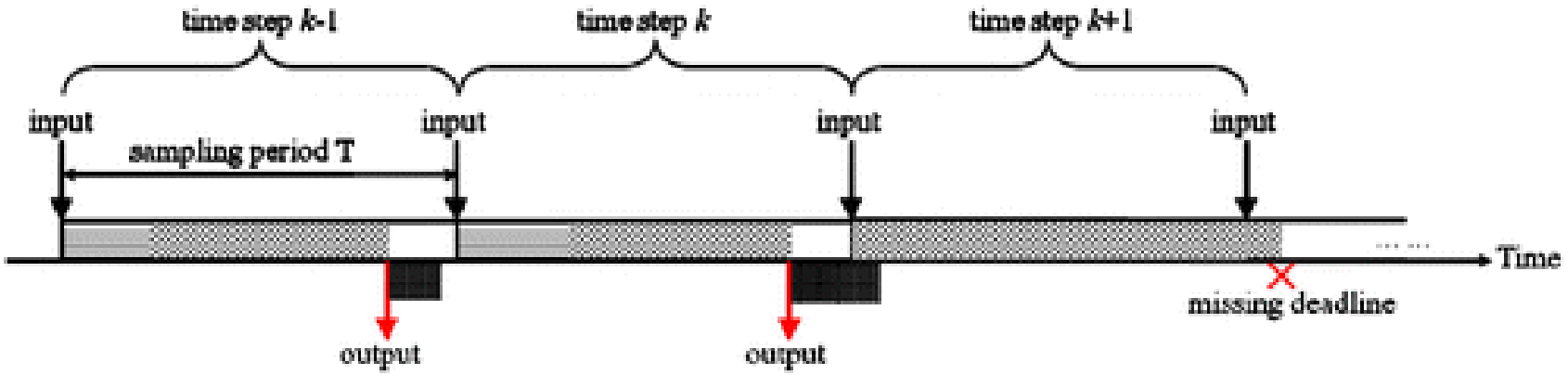
Real-time rendering

- Time-budget (time-critical) rendering
  - Rendering in a fixed budget of time



Real-time rendering

- Time-budget (time-critical) rendering
  - Rendering in a fixed budget of time



- Rendering very fast
  - Traditional criteria: 30 FPS
  - Now for VR: 90 FPS

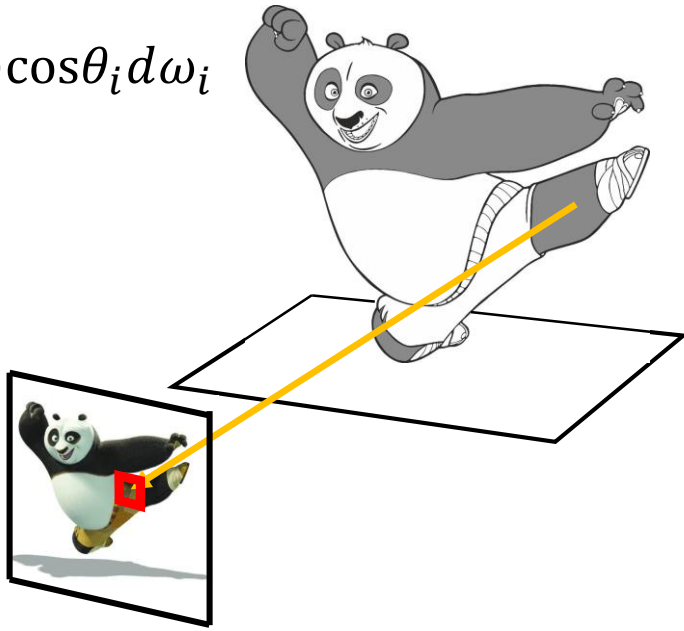
Real-time rendering

- Simplifying the computation of rendering equation

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

$$= L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

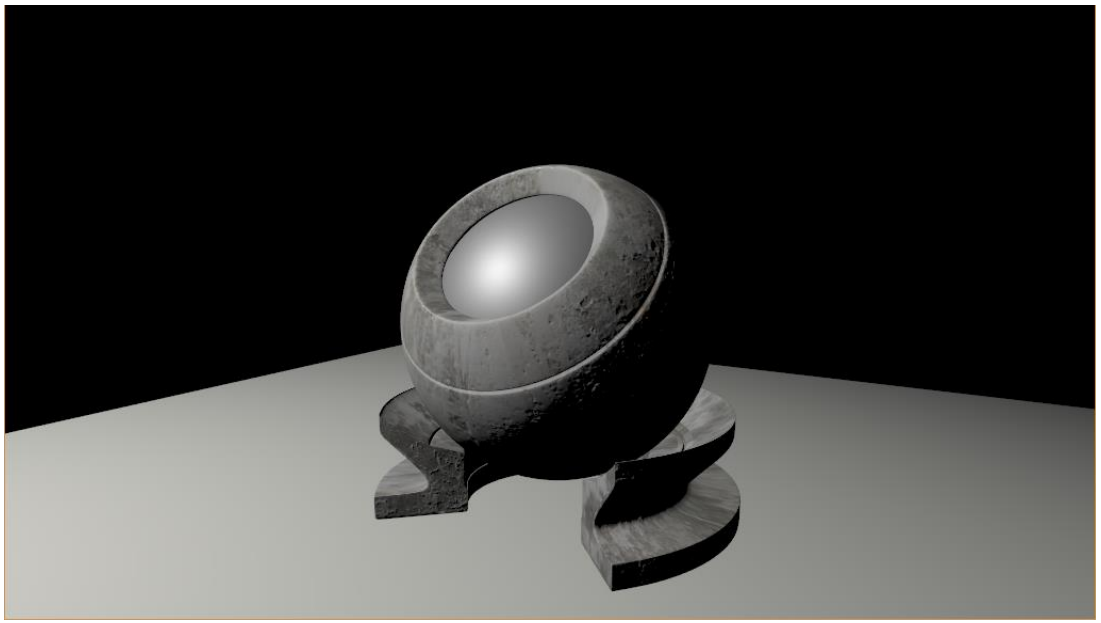
$$= L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

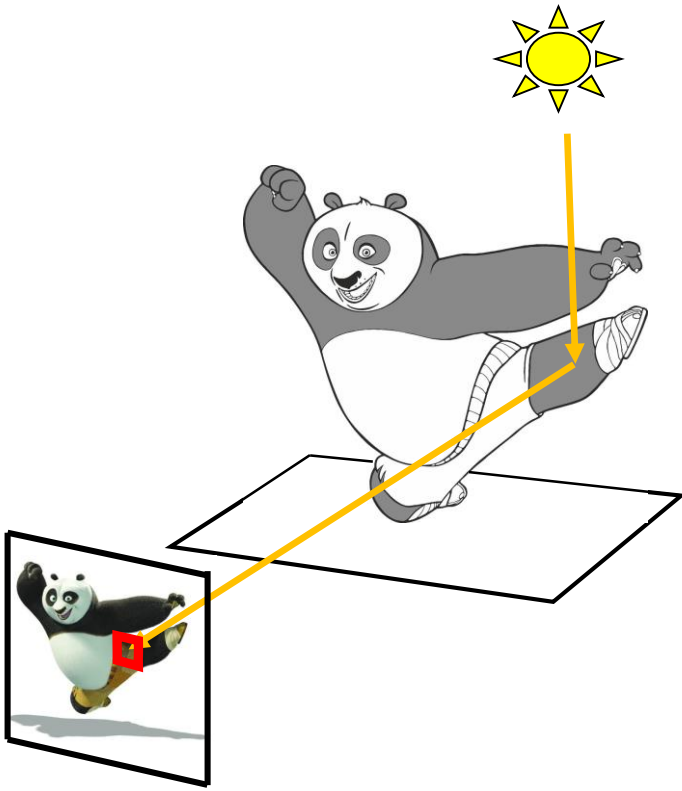


Real-time rendering

- Point light local shading

$$L_o(x, \omega_o) = L_i(x, \omega_i) f_r(x, \omega_i \rightarrow \omega_o) \cos\theta_i$$

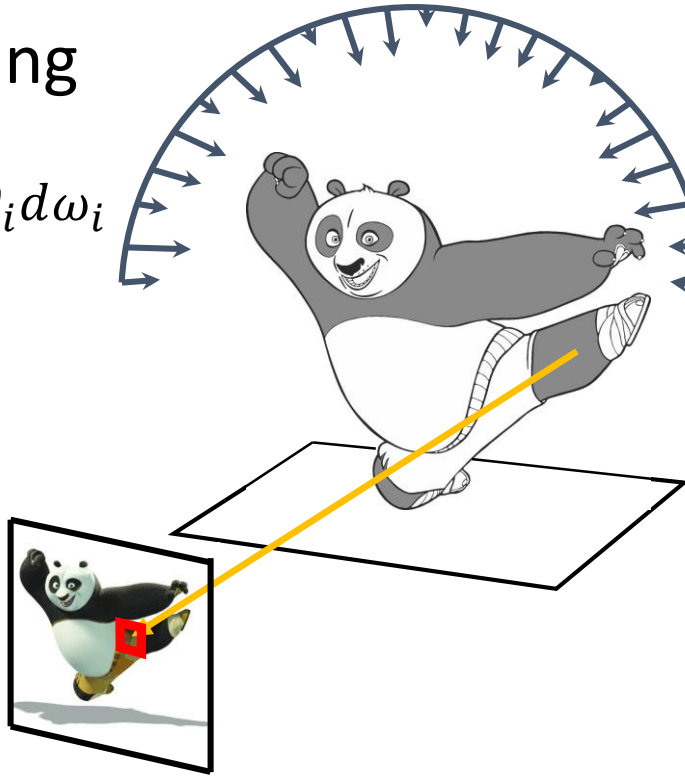




Real-time rendering

- Environment/Area light local shading

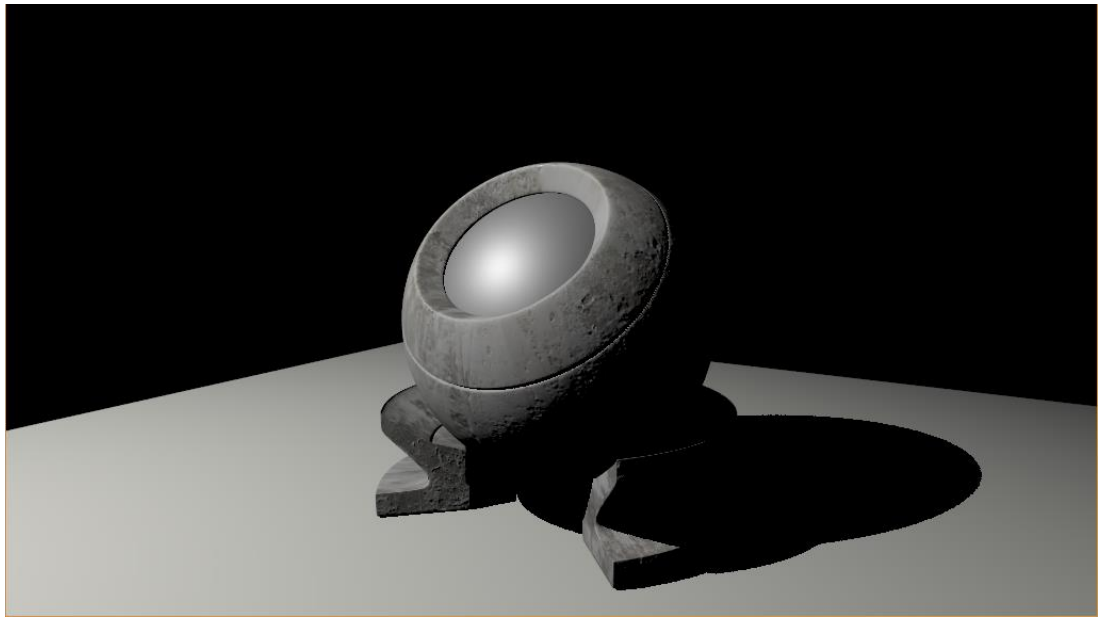
$$L_o(x, \omega_o) = \int_{H^2} L_i(x, \omega_i) f_r(x, \omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

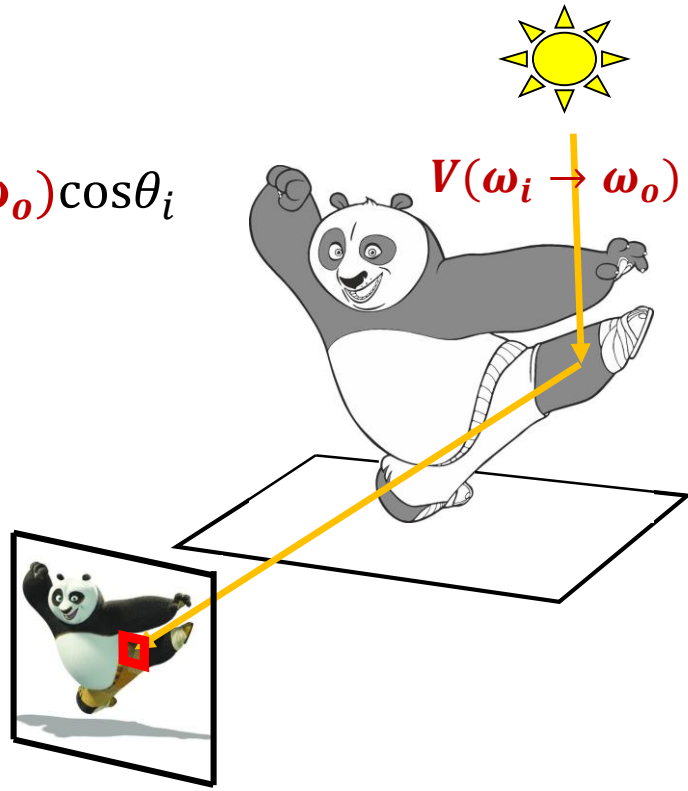


Real-time rendering

- Point light local shading + Shadow

$$L_o(x,\omega_o) = L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \boldsymbol{V(\omega_i \rightarrow \omega_o)} \cos\theta_i$$





Real-time rendering

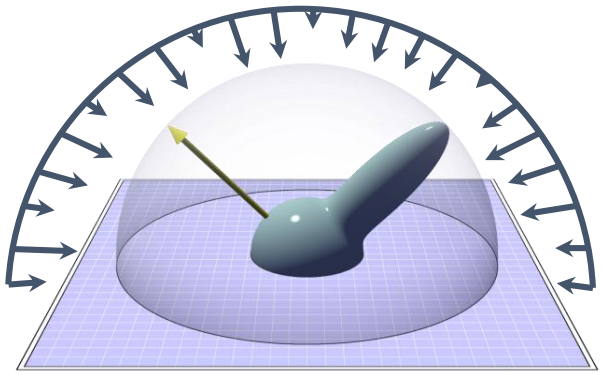
- Environment/Area light local shading + Shadow

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \boldsymbol{V(\omega_i \rightarrow \omega_o)} \cos\theta_i d\omega_i$$

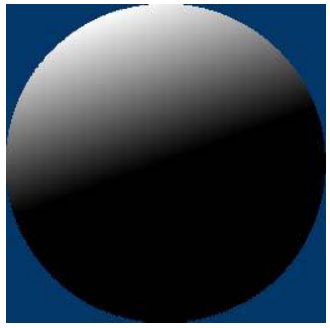
Real-time rendering

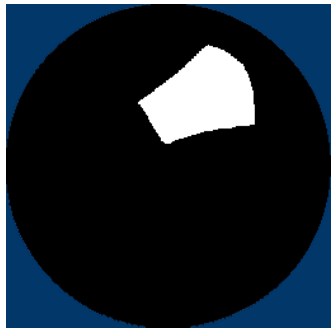
- Environment/Area light local shading + Shadow

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) \boldsymbol{V(\omega_i \rightarrow \omega_o)} \cos\theta_i d\omega_i$$



$\int$



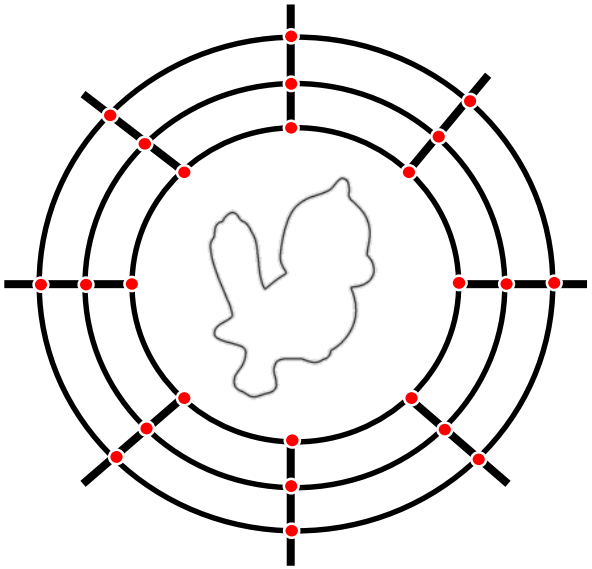


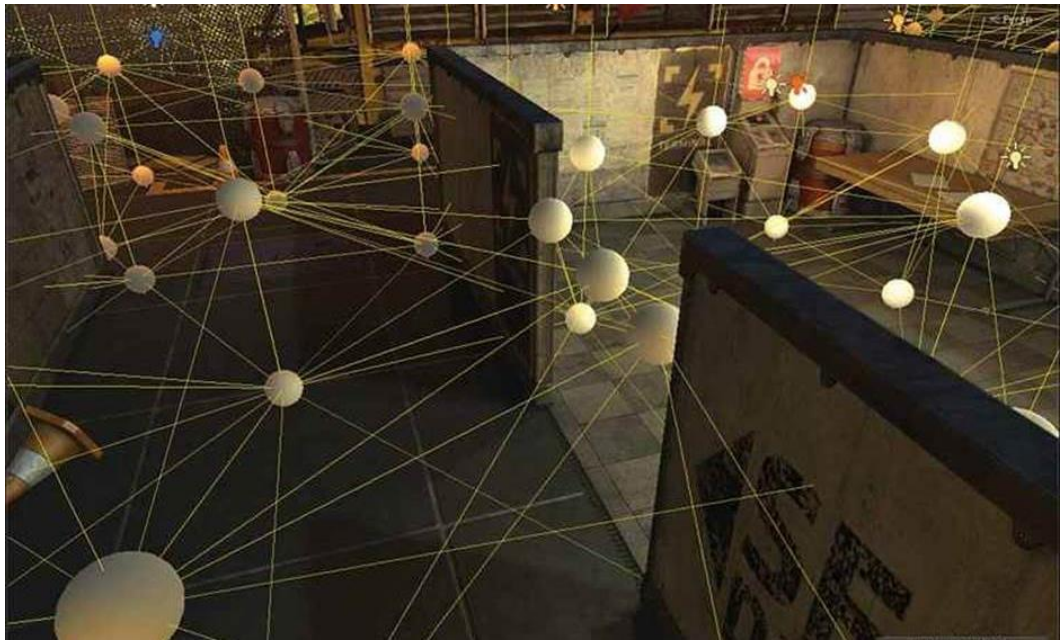
Real-time rendering

- Environment/Area light local shading + Shadow

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \to \omega_o) V(\omega_i \to \omega_o) \cos\theta_i d\omega_i$$

  - Approximate incident radiance
    - Probes, Fields, etc.



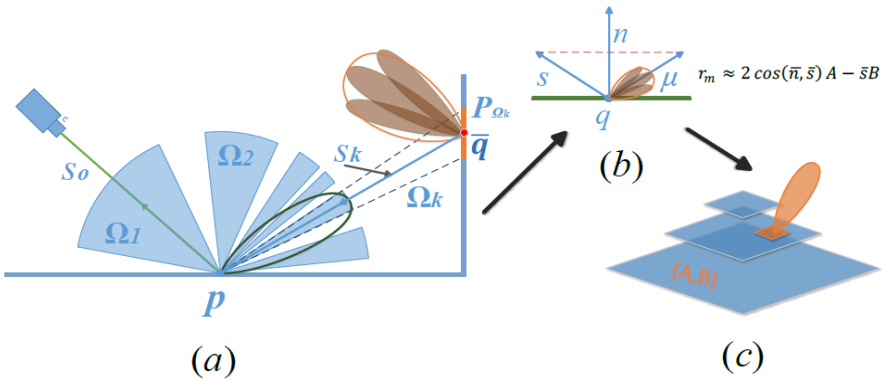


Real-time rendering

- Environment/Area light local shading + Shadow

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) V(\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

  - Approximate BRDFs
    - Fitting, MIP-Mapping, etc.

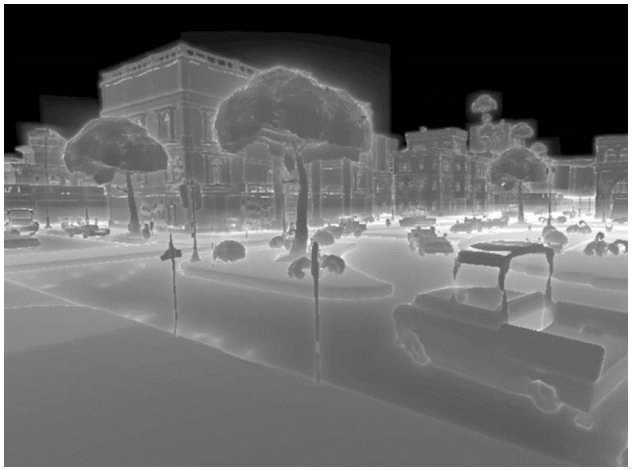


Real-time rendering

- Environment/Area light local shading + Shadow

$$L_o(x,\omega_o) = \int_{H^2} L_i(x,\omega_i) f_r(x,\omega_i \rightarrow \omega_o) V(\omega_i \rightarrow \omega_o) \cos\theta_i d\omega_i$$

  - Approximate Visibilities
    - AO, basis functions, simple occluders, contours, distance field, etc.



Rendering Tutorial IV:
- Precomputed Radiance Transfer

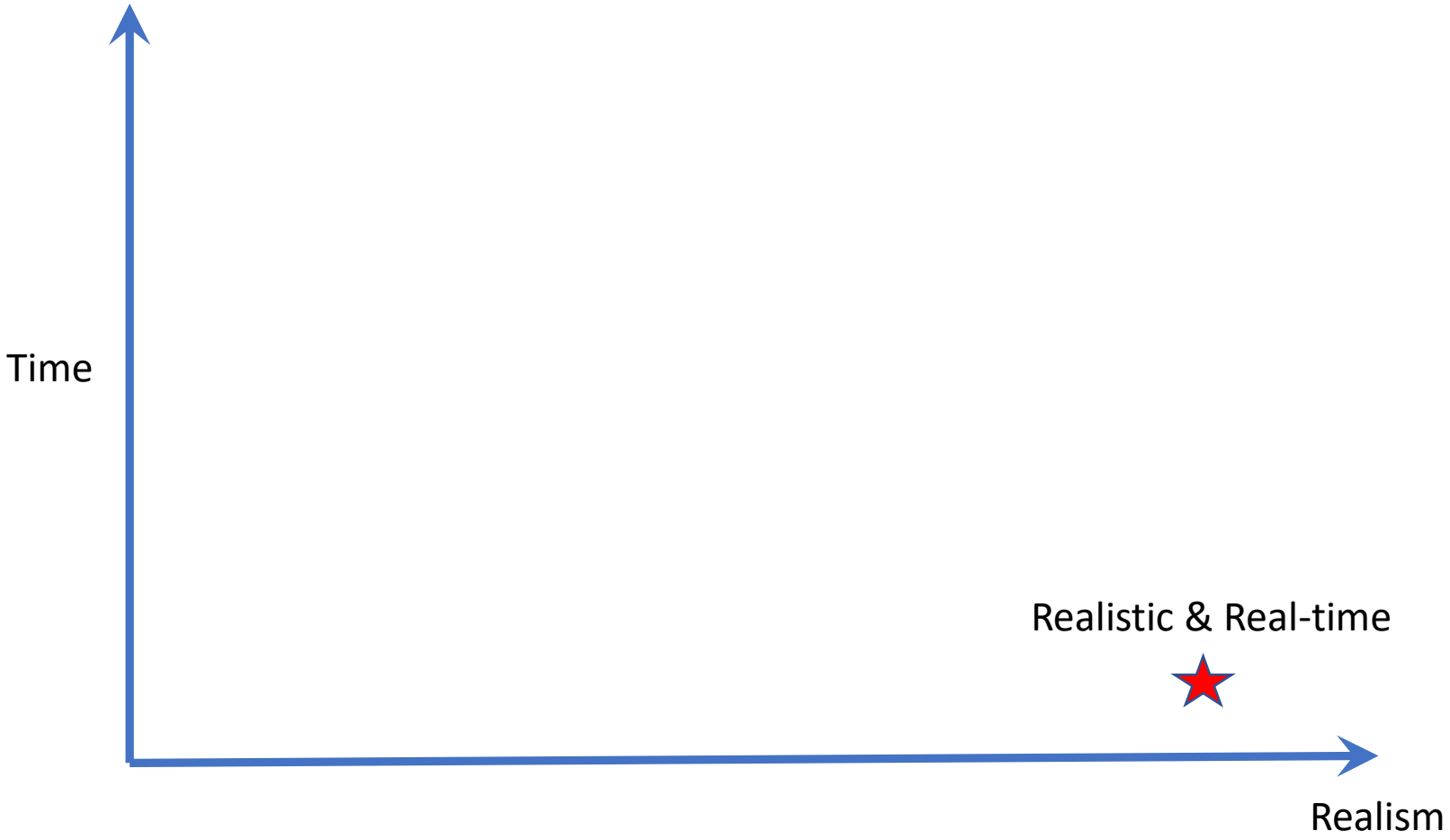
Abstract

本报告作为一个4节Tutorial系列的第四个报告，将介绍实时绘制中的一个重要技术--预计算辐射传（Precomputed Radiance Transfer, PRT）。PRT技术在实时绘制、游戏领域应用广泛。PRT的基本思想是：通过预计算，将绘制积分中的光源函数、辐射传输函数（或可见性函数和BRDF）投影到某一基函数空间，然后将绘制积分转化为基函数空间的点积运算。本报告将介绍PRT的背景和基本概念、基函数选择和多种绘制应用（静态场景、动态场景、半透明绘制、毛发绘制）。此外，本报告还将介绍绘制积分的解析近似方法，这类方法的思路与PRT类似，但无需预计算，并具有动态性的优点。

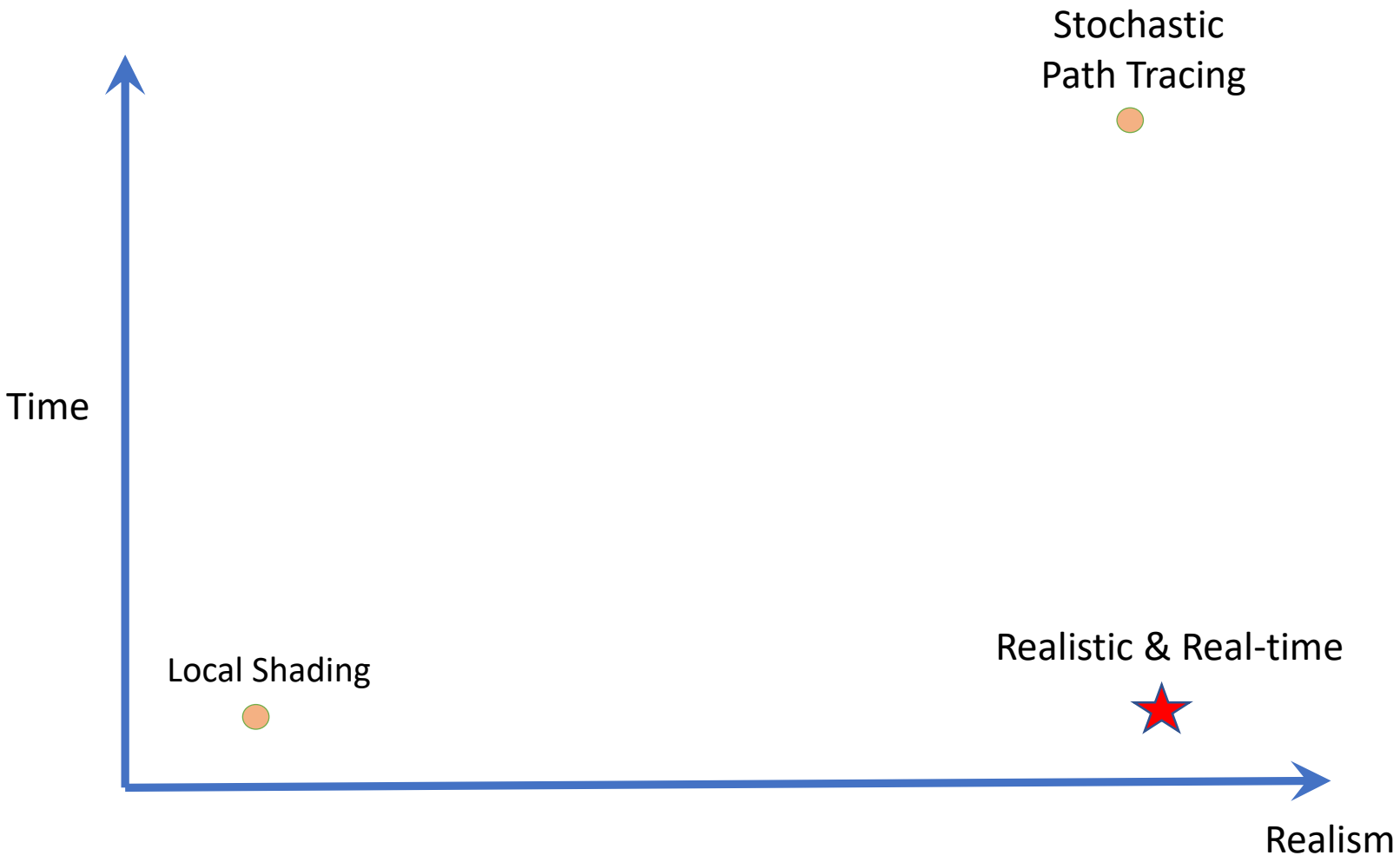
Bio

徐昆，清华大学计算机系教研系列副教授，博士生导师。2009年毕业于清华大学获博士学位。研究方向为计算机图形学，主要从事真实感绘制、可视媒体内容编辑与生成等方面的研究。发表SCI论文20余篇，其中12篇论文发表在ACM TOG, IEEE TVCG等重要期刊和会议上。曾获国家自然科学奖二等奖（排名第4），中国计算机学会优秀博士学位论文奖，入选中国科协“青年人才托举工程”。

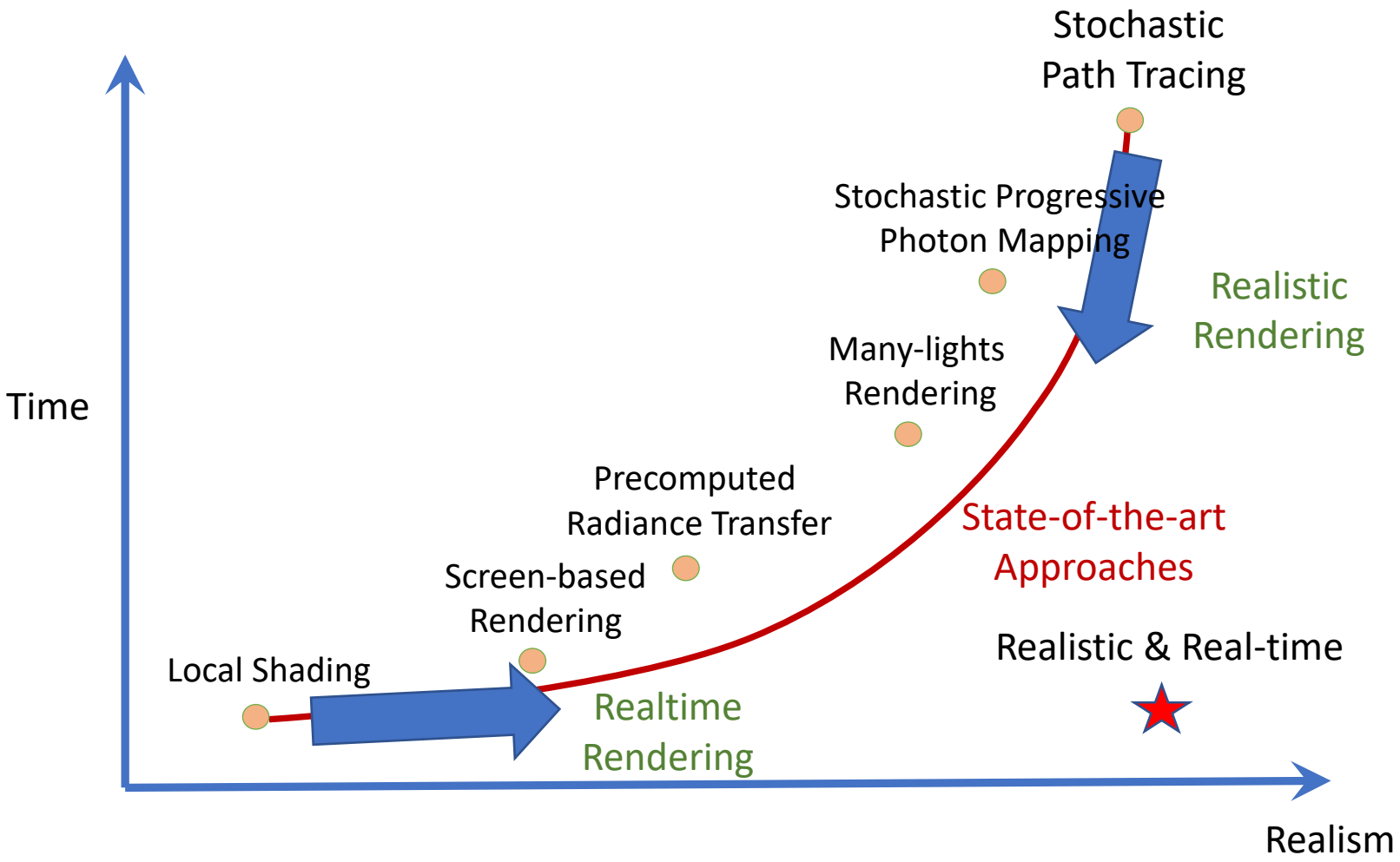
Goal of Rendering



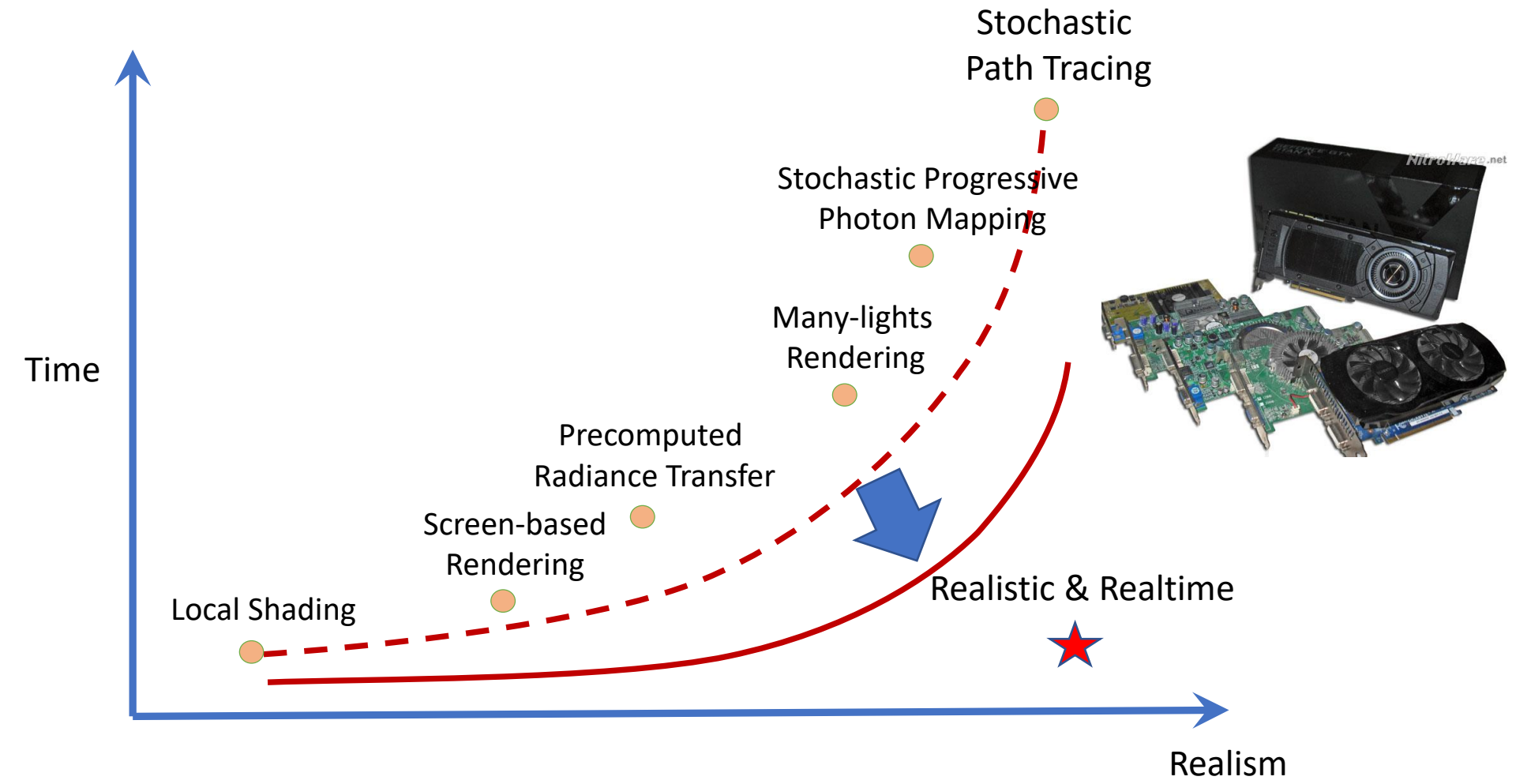
Goal of Rendering



Goal of Rendering

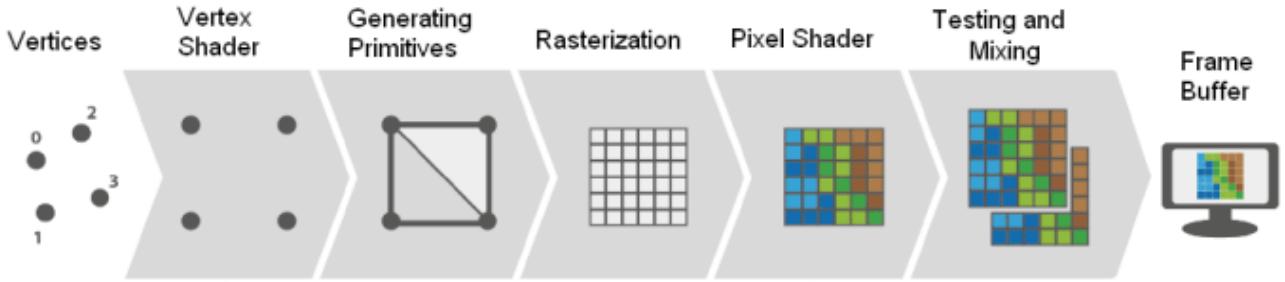


Another Dimension to Achieve Our Goal

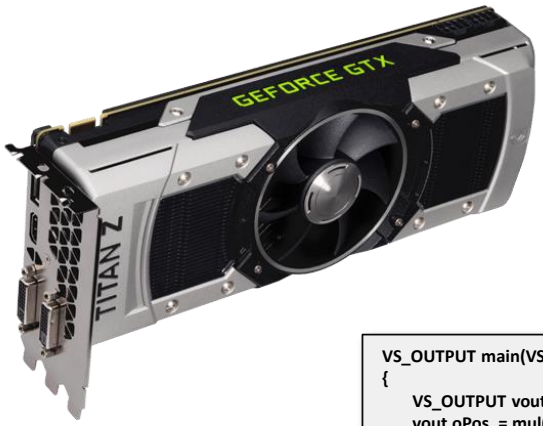


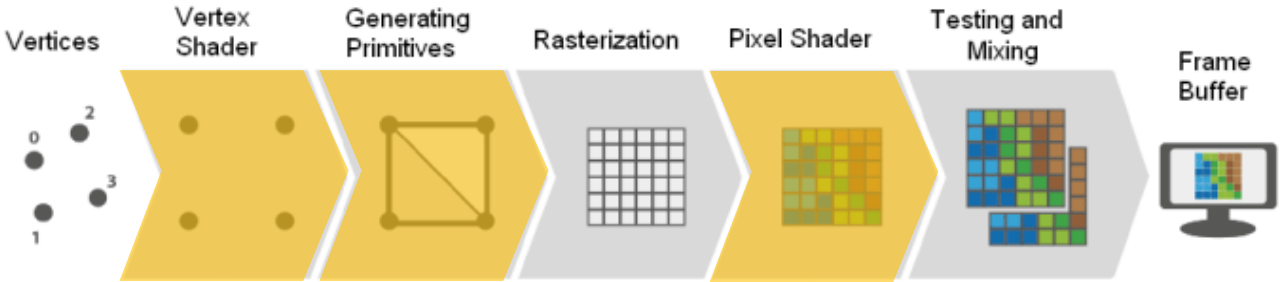
Hardware-based Rendering Pipeline

Hardware-based Rendering Pipeline



Hardware-based Rendering Pipeline



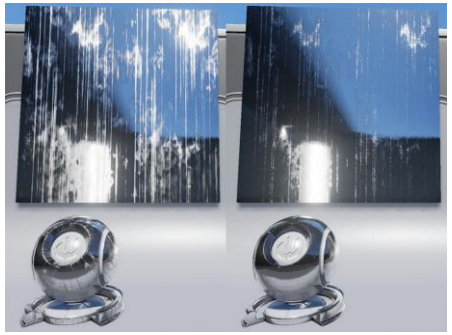



```
VS_OUTPUT main(VS_INPUT in)
{
    VS_OUTPUT vout;
    vout.oPos  = mul(wVP, invPos);
    vout.wPos = mul(wM, in.vPos).xyz;
    return vout;
}
```

```
PS_OUTPUT main(VS_OUTPUT in)
{
    PS_OUTPUT pout;
    float3 N = normalize(in.N);
    float3 P = in.wPos.xyz;
    float3 L = normalize(lightPos – P);
    pout.diffuse = Kd * Ic* dot(N,L);
    return pout;
}
```

```
GS_OUTPUT main(VS_INPUT in)
{
    GS_OUTPUT out;
    out.oPos  = mul(wVP, invPos);
    float3 N = normalize(in.N);
    float3 P = in.wPos.xyz;
    return out;
}
```

models

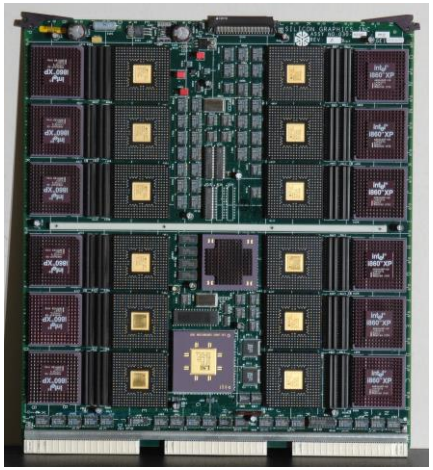


Reflection

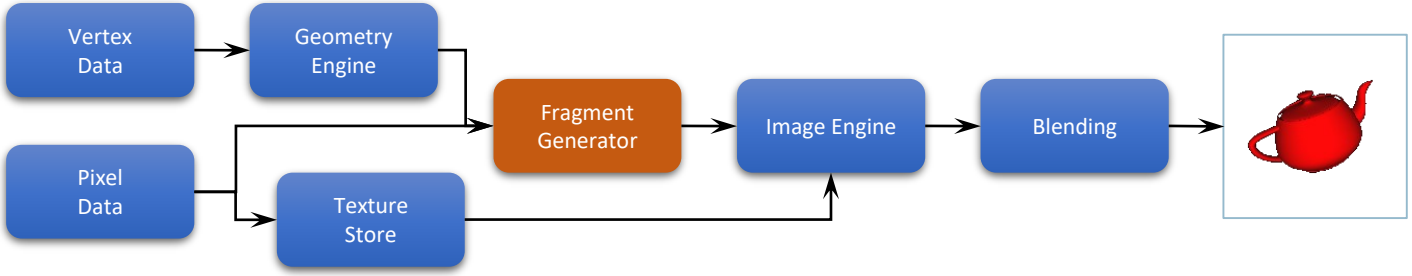
Resolution scale

Shaft

In the Beginning …



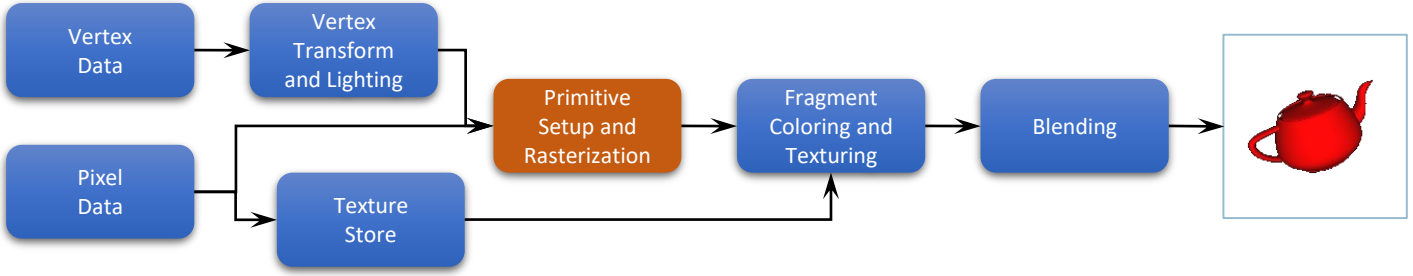
- IrisGL is a proprietary graphics API created by Silicon Graphics (SGI) in the early 1980s
- SGI opened source a version of IrisGL as a public standard called **OpenGL**.



- In 1992, SGI led the creation of the OpenGL Architecture Review Board (OpenGL ARB)

Beginning of OpenGL

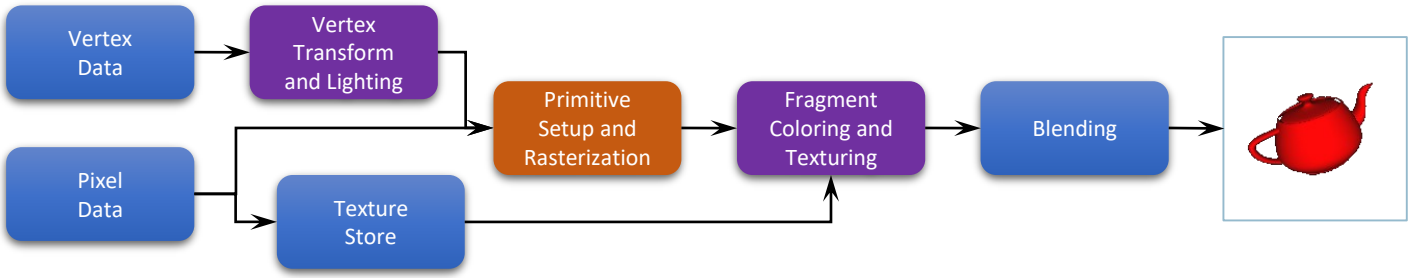
- OpenGL 1.0 was released on July 1st, 1994
- Its pipeline was entirely *fixed-function*
  - the only operations available were fixed by the implementation



- The pipeline evolved
  - but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0 (Sept. 2004)

Beginnings of The Programmable Pipeline

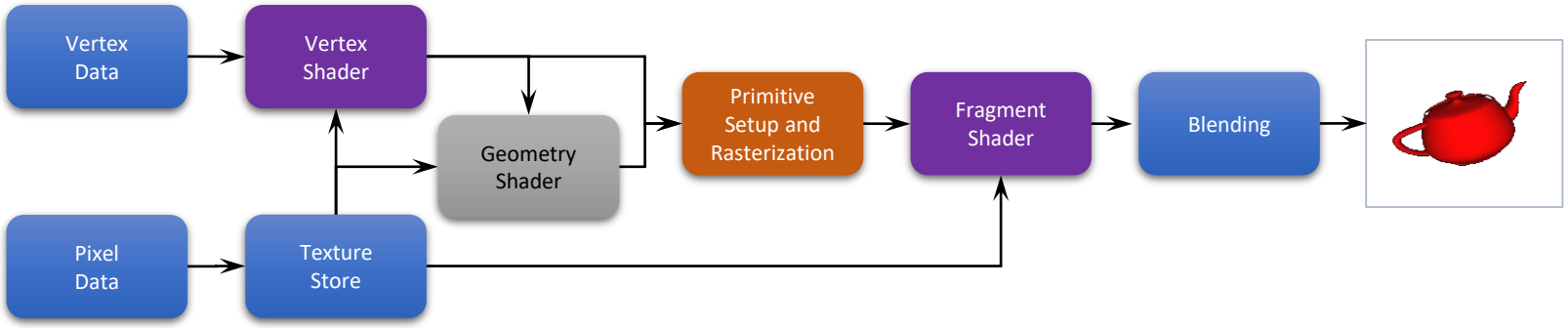
- OpenGL 2.0 (officially) added programmable shaders
  - *vertex shading* augmented the fixed-function transform and lighting stage
  - *fragment shading* augmented the fragment coloring stage



- However, the fixed-function pipeline was still available

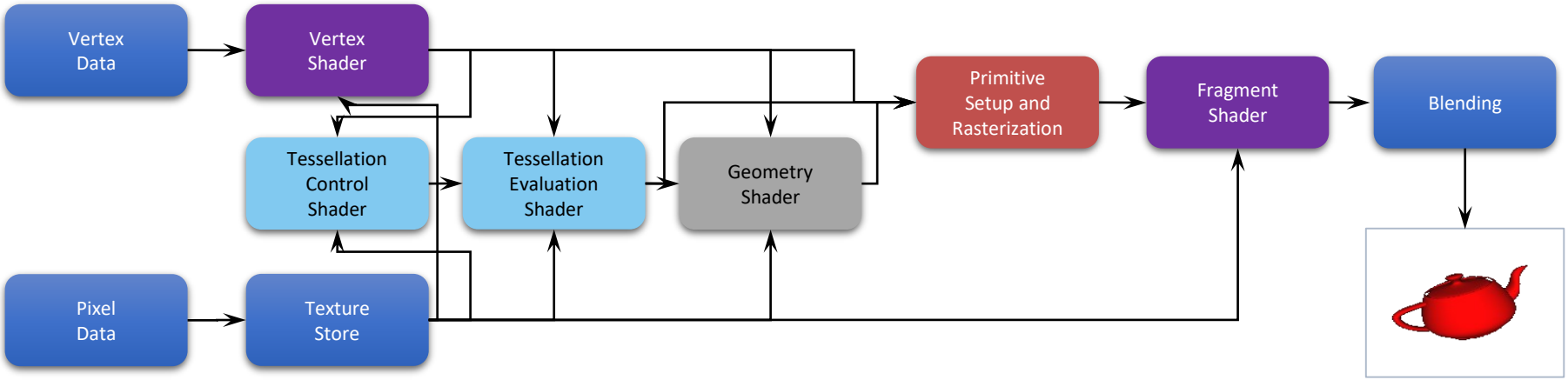
More Programmability

- OpenGL 3.2 (released August 3rd, 2009) added an additional shading stage – geometry shaders
  - modify geometric primitives within the graphics pipeline



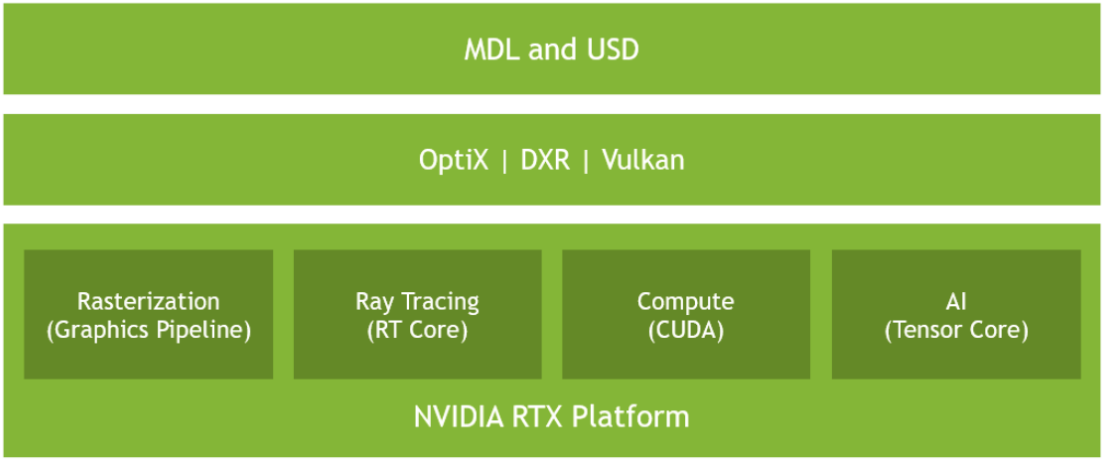
More and More Programmability

- OpenGL 4.1 (released July 25th, 2010) included additional shading stages – *tessellation-control* and *tessellation-evaluation* shaders
- Latest version is 4.6 (released July 31th, 2017)



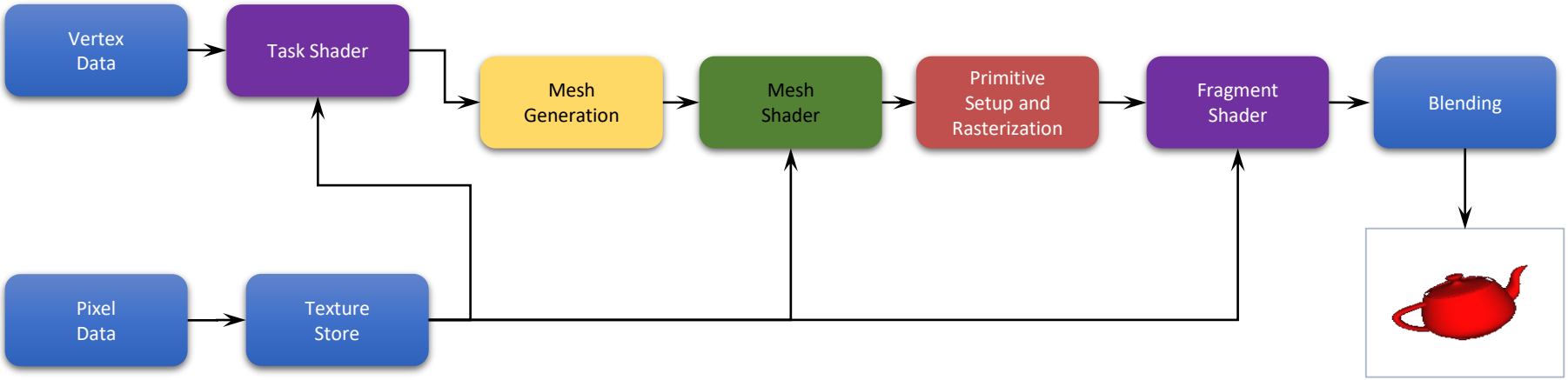
The Latest Pipelines

- August 20th, 2018, nVidia released Geforce RTX, which is based on Turing microarchitecture and features real-time ray tracing.



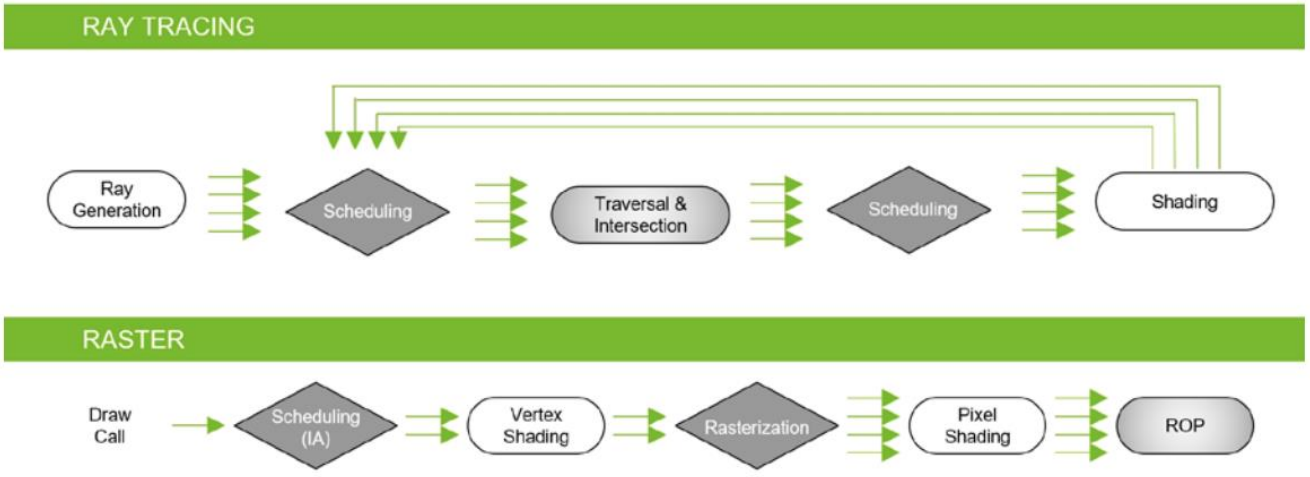
The Latest Pipelines

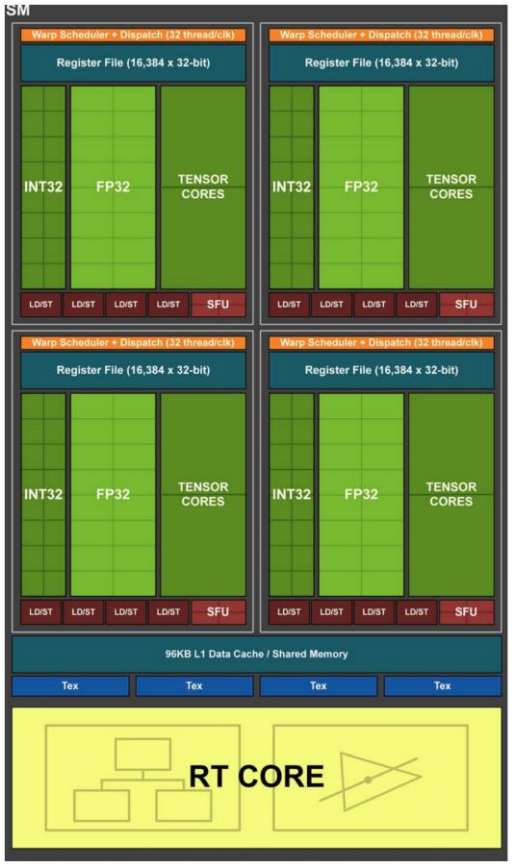
- August 20th, 2018, nVidia released Geforce RTX, which is based on Turing microarchitecture and features real-time ray tracing.
- New Mesh Shader



The Latest Pipelines

- August 20th, 2018, nVidia released Geforce RTX, which is based on Turing microarchitecture and features real-time ray tracing.
- New Mesh Shader
- New RT Core





Latest Graphics APIs

- OpenGL, Vulkan, D3D12, and Metal
  - Coming to platforms you care about
- Reduce CPU overhead/bottlenecks
- More stable/predictable driver performance
- Explicit, console-like control

Ray Tracing APIs on Graphics Hardware

- DirectX Raytracing (DXR)
- nVidia RTX ray tracing APIs (OptiX APIs, 5.1 latest)
- Ray-tracing extension to the Vulkan

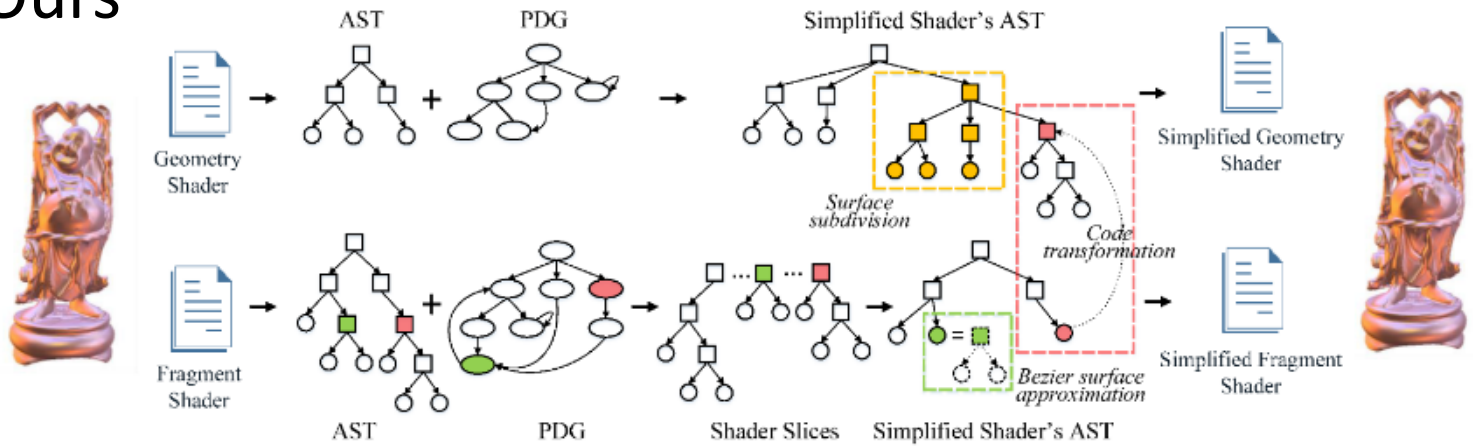
What research can we do with rapidly evolved hardware?

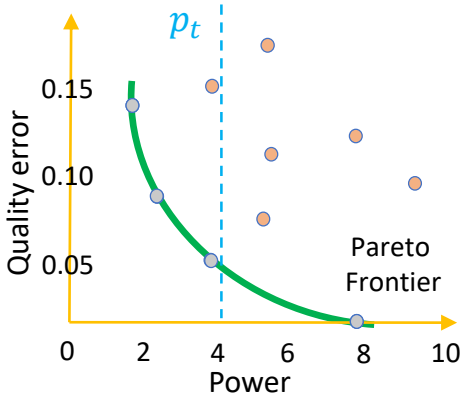
- New algorithms on new hardware.
- Pipeline optimization

Pipeline Optimization

- Ours

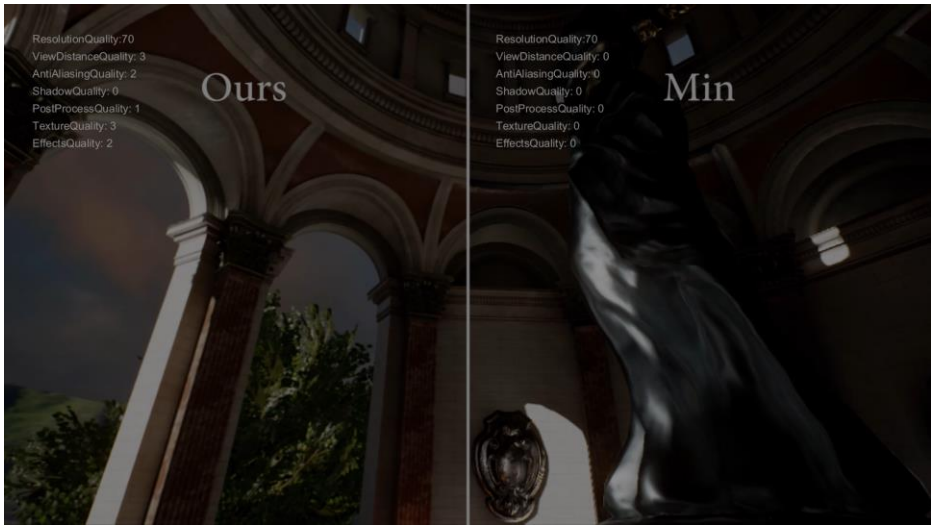
Shader optimization， 2013





$$\text{argmin}_{s_i}\ \text{e}(f(s_i))$$
$$\text{s.t.}\ \ p(f(s_i)) < p_t$$

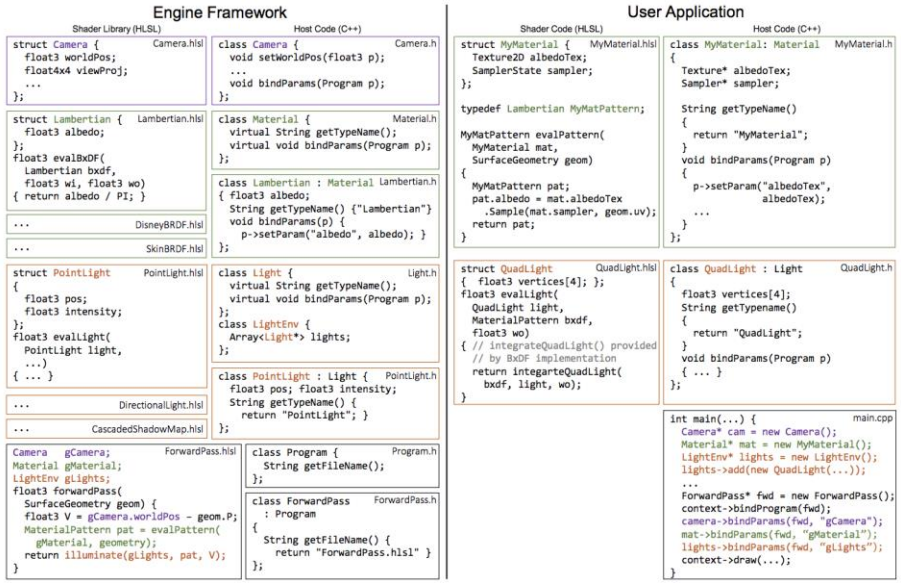
Power optimization, 2016



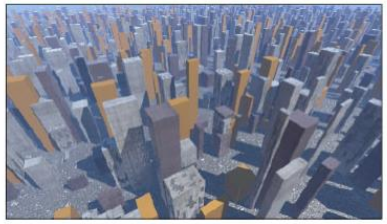
Pipeline Optimization

Slang, 2018

- He et al.'s work



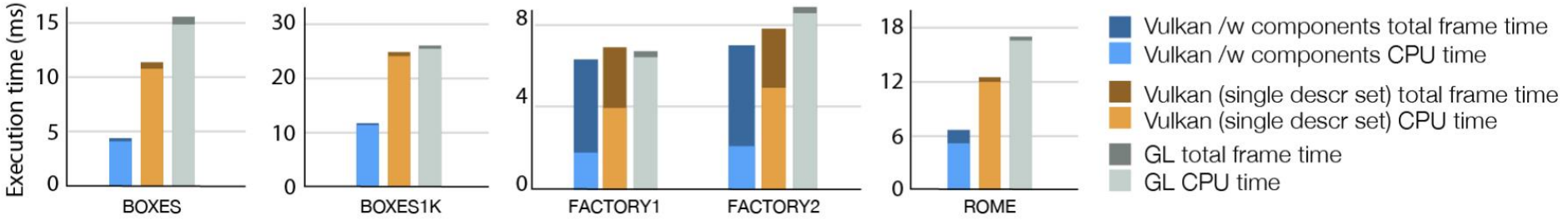
Shader components, 2017


BOXES / BOXES1K


FACTORY1


FACTORY2


ROME



Conclusion

Takeaway

- Rendering equation
- Solving rendering equation
- Hardware evolution for rendering

Not Covered in This Talk

- Volumetric / Participating Media Rendering
  - RTE, sampling, reconstruction, etc.
- Rendering Specific Objects
  - Tree, grass, water, human, etc.
- Rendering Large-scale Scenes
  - Out-of-core data management, visibility culling.
- Screen Space Post-processing Technique
  - Tone mapping, SSR, SSAO, blooming, DOF, etc.
- Rendering Engine Frameworks / Architectures

Thank you!
Q & A