GEOMETRY INDEPENDENT SURFACE LIGHT FIELDS FOR REAL TIME RENDERING OF PRECOMPUTED GLOBAL ILLUMINATION

Ehsan Miandji and Joel Kronander and Jonas Unger

Computer Graphics and Image Processing Group
Linköping University

THE GOAL

- Real Time Rendering of globally illuminated scenes:

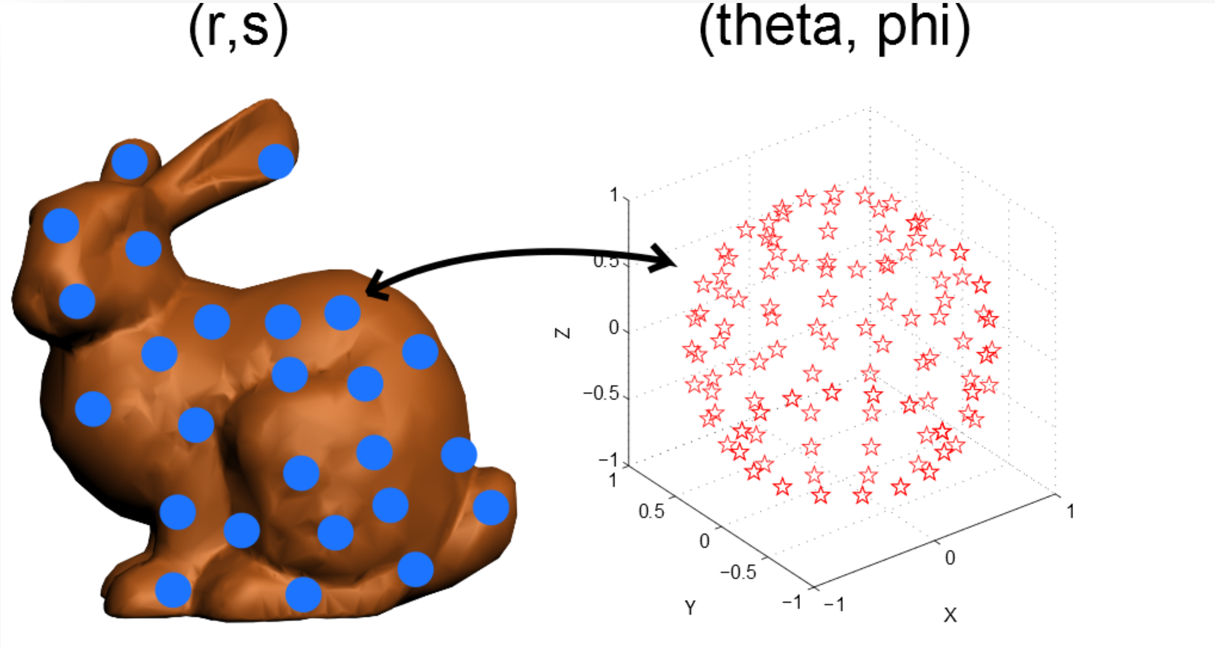
THE GOAL

- Real Time Rendering of globally illuminated scenes:

- Assumptions:
  - Free viewpoint
  - Static scenes
  - Fixed lighting
  - All-frequency

BACKGROUND

- Surface Light Field (SLF): $f(r,s,\theta,\varphi)$



[MRP98]

CONTRIBUTIONS

- Geometry independent SLF
- Application of CPCA for SLF compression
- Real time rendering of CPCA compressed data
- A fast power iteration method

Method Overview

Data Generation

1. SLF function discretization
   1. Spatial sampling
   2. Angular sampling
2. Compute outgoing radiance
3. Store results in SLF Matrix *F*

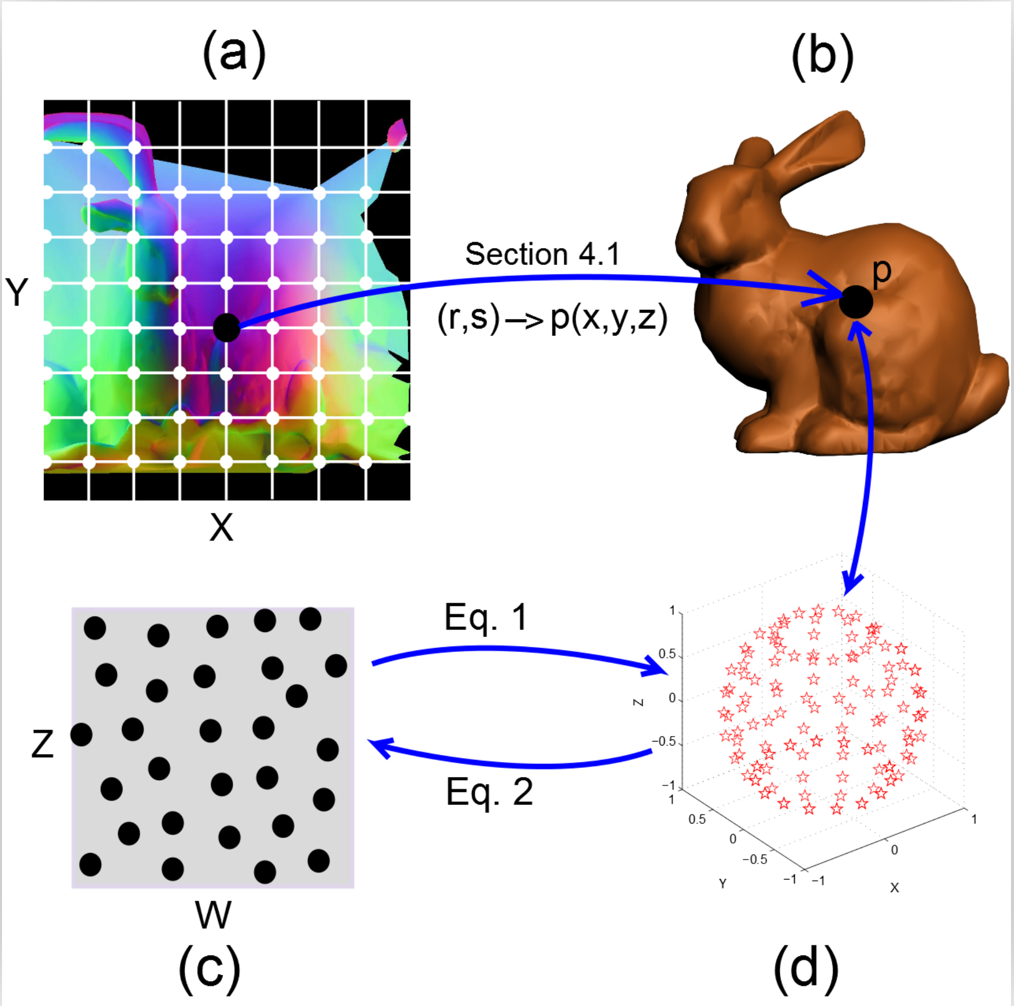
Compression (CPCA)

1. Cluster *F*
2. Apply PCA on each cluster

Rendering

1. For each ray/view direction
2. Find the corresponding surface point and direction
3. Fetch data and decode

Data Generation



DATA GENERATION

- The result is a m x n matrix:

$$F = \begin{bmatrix} f(r_1, s_1, \theta_1, \varphi_1) & . & . & . & f(r_1, s_1, \theta_Z, \varphi_W) \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ f(r_X, s_Y, \theta_1, \varphi_1) & . & . & . & f(r_X, s_Y, \theta_Z, \varphi_W) \end{bmatrix}$$

Compression

- Huge amount of data (typically +1.5GB per shape)
- What we need is a method with:
  - High compression ratio
  - Faithful reconstruction of data
  - Random access
- CPCA vs. PCA

Clustered Principal Component Analysis (CPCA)

1. Mean subtraction

$$G = [x_{p1} - \mu_{p2}, x_{p2} - \mu_{p1}, \ldots, x_{pm} - \mu_{pm}]^T$$

2. Clustering
3. PCA on clusters

$$G_i = U_i V_i^T$$

4. Surface and view maps

Clustered Principal Component Analysis (CPCA)

- We use power iteration for PCA
  - Calculate few terms (not full SVD)
  - Progressive
  - Low memory footprint
- Not efficient for CPCA
- Solution: Modified Power Iteration
- If $m > n$
  - *Compute eigenvectors and eigenvalues of* $A = F^T F$
- Else
  - *Compute eigenvectors and eigenvalues of* $A = F F^T$

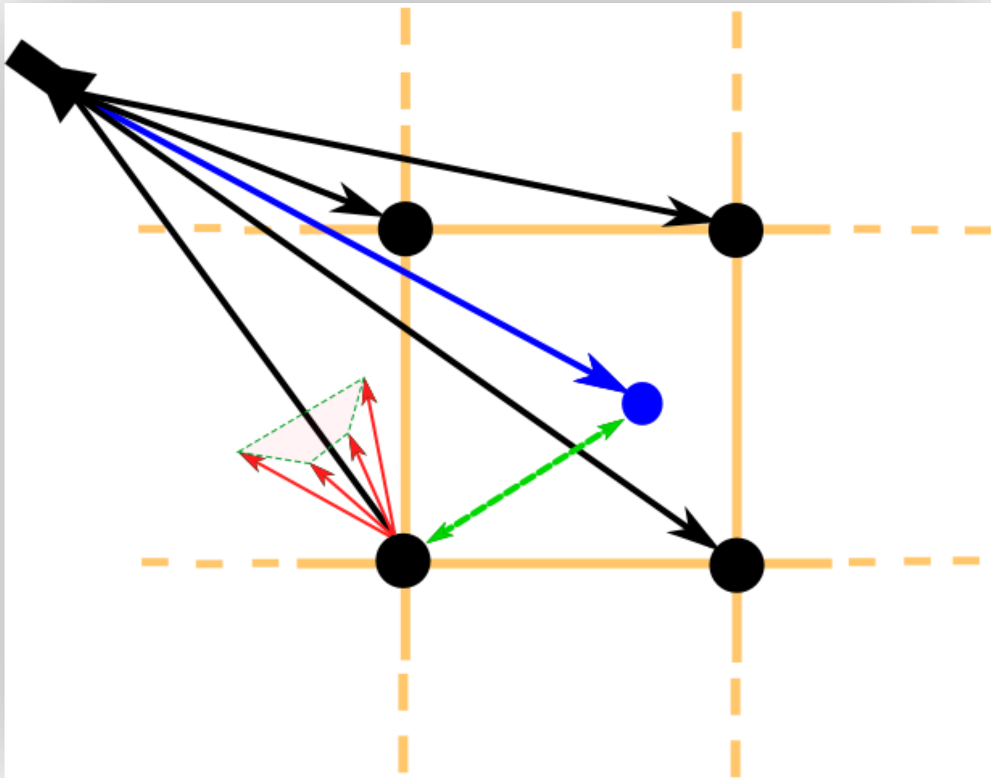
RENDERING

$$f(r,s,\theta,\varphi) \approx F[r,s,u,v] = \left(\sum_{j=1}^{k} U_i[\alpha, j] V_i[\beta, j]^T \right) + \mu[\alpha]$$

- Single pass rendering
- We have
  - Intersection point (and the texture coordinate of it)
  - View vector
- We need
  - Surface and view map addresses (α and β)

RENDERING

- Reconstruction and Interpolation

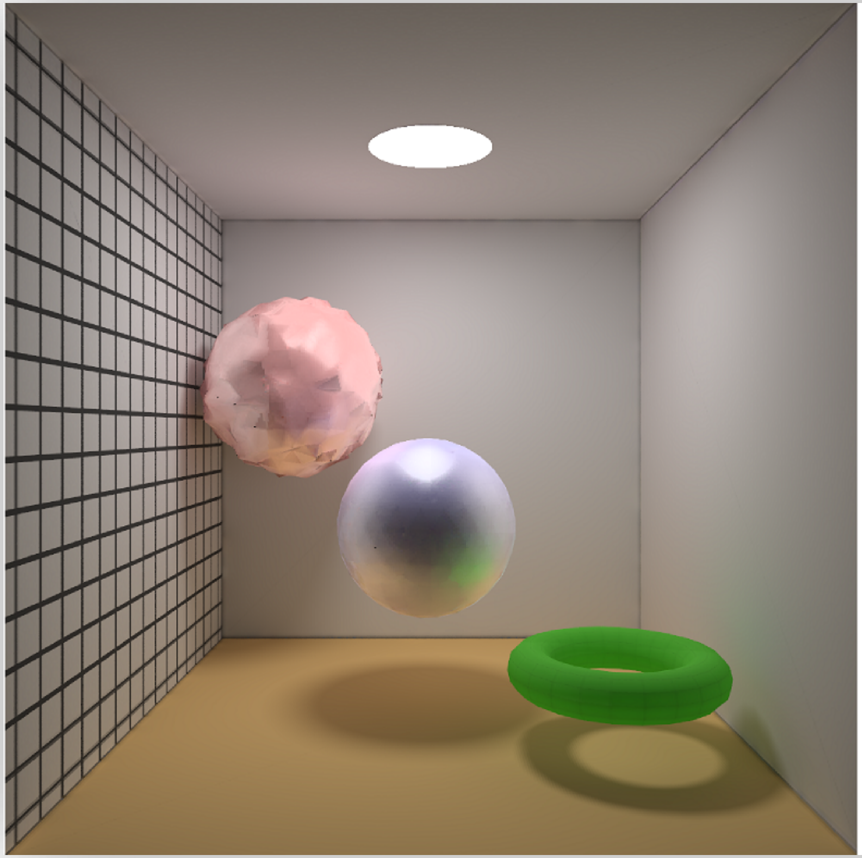


IMPLEMENTATION

- Data Generation
  - PBRT (as a renderer)
  - Different surface integrators can be used
- Compression
  - K-Means
  - Hand tuned PCA!
  - Guaranteed convergence
- Renderer
  - CPU based (PBRT)
  - GPU based (DirectX + HLSL)

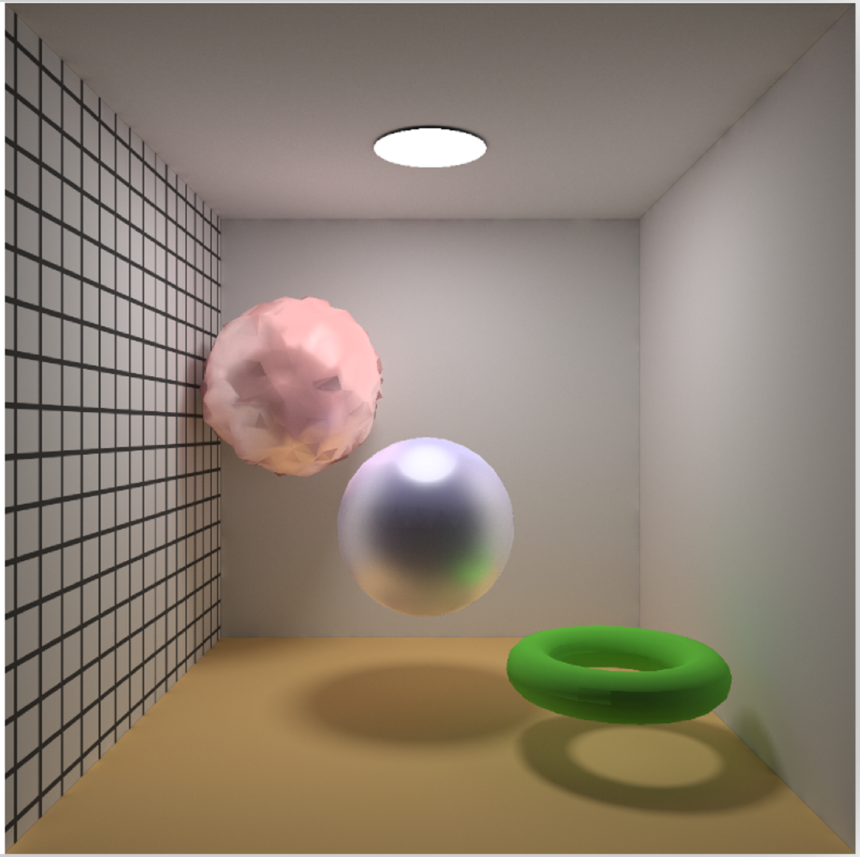
Results (CPU)

Our method



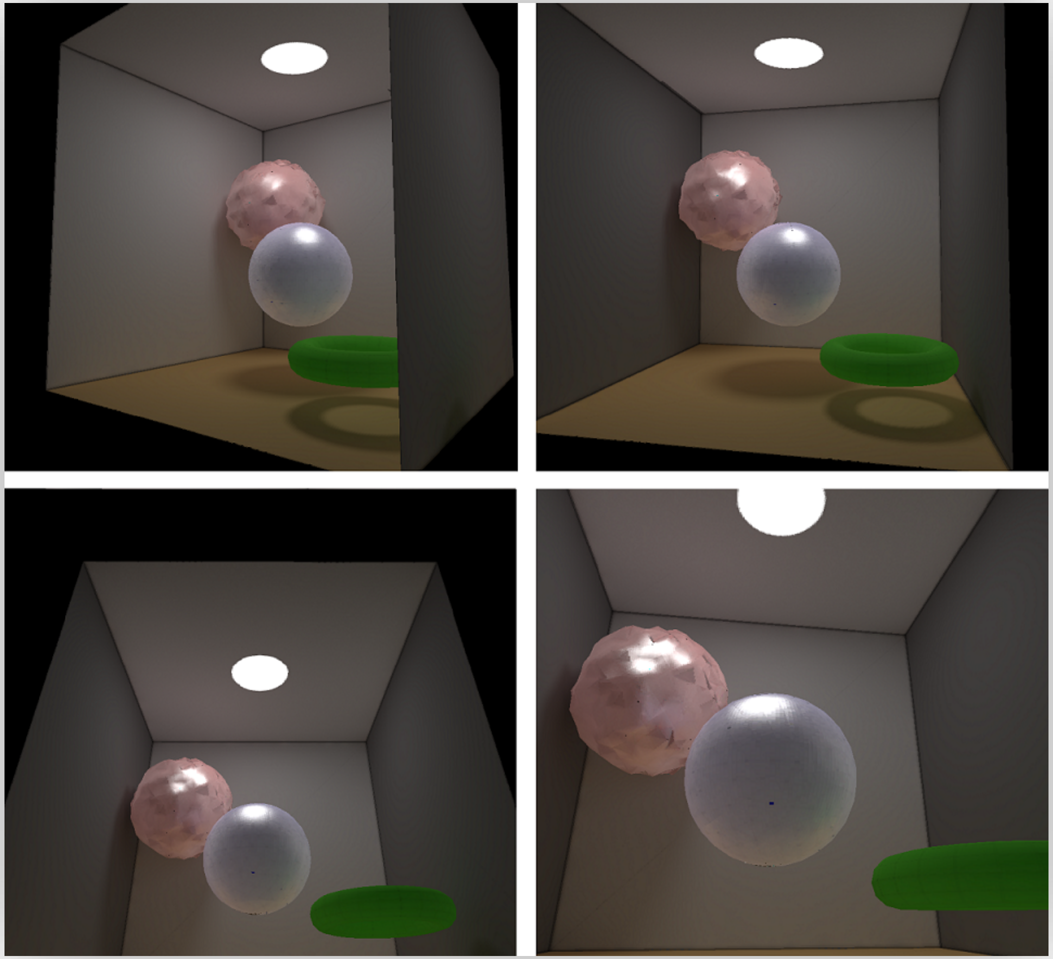
16.7 sec

Reference



202 min

RESULTS (GPU)



~120 Frames Per Second

THANK YOU

vcl.itn.liu.se