POINT BASED COLOR BLEEDING WITH CUDA AND CACHING

Nick Feeney

Outline

- Introduction
- Background
- Related Work
- Current Implementation
- Results
- Future Work
- Conclusion

Introduction

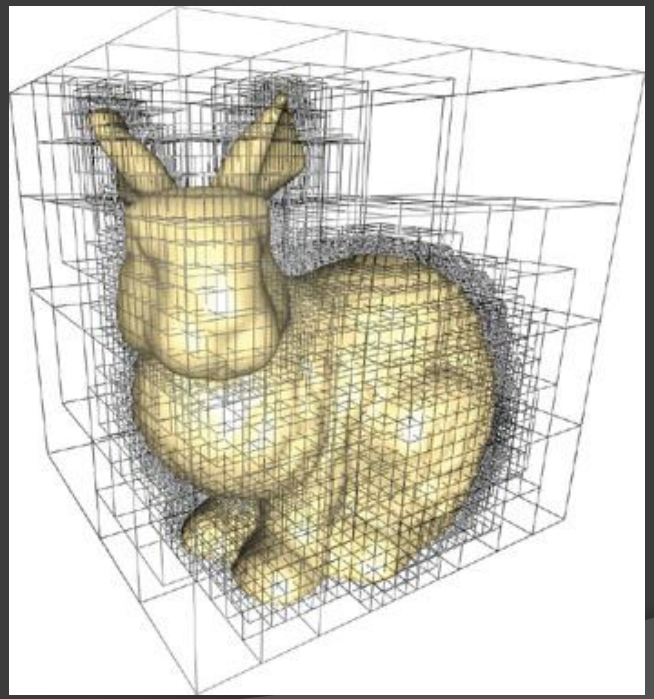
- Reasons for Project:
  - Global Illumination gives up high quality and realistic images.
  - Global Illumination is a lot of work and can always go fast.

Introduction

- Project central goals:
  - Implementing a Point Based Color Bleeding global illumination algorithm.
  - To speed up with CUDA
  - To speed up with Caching

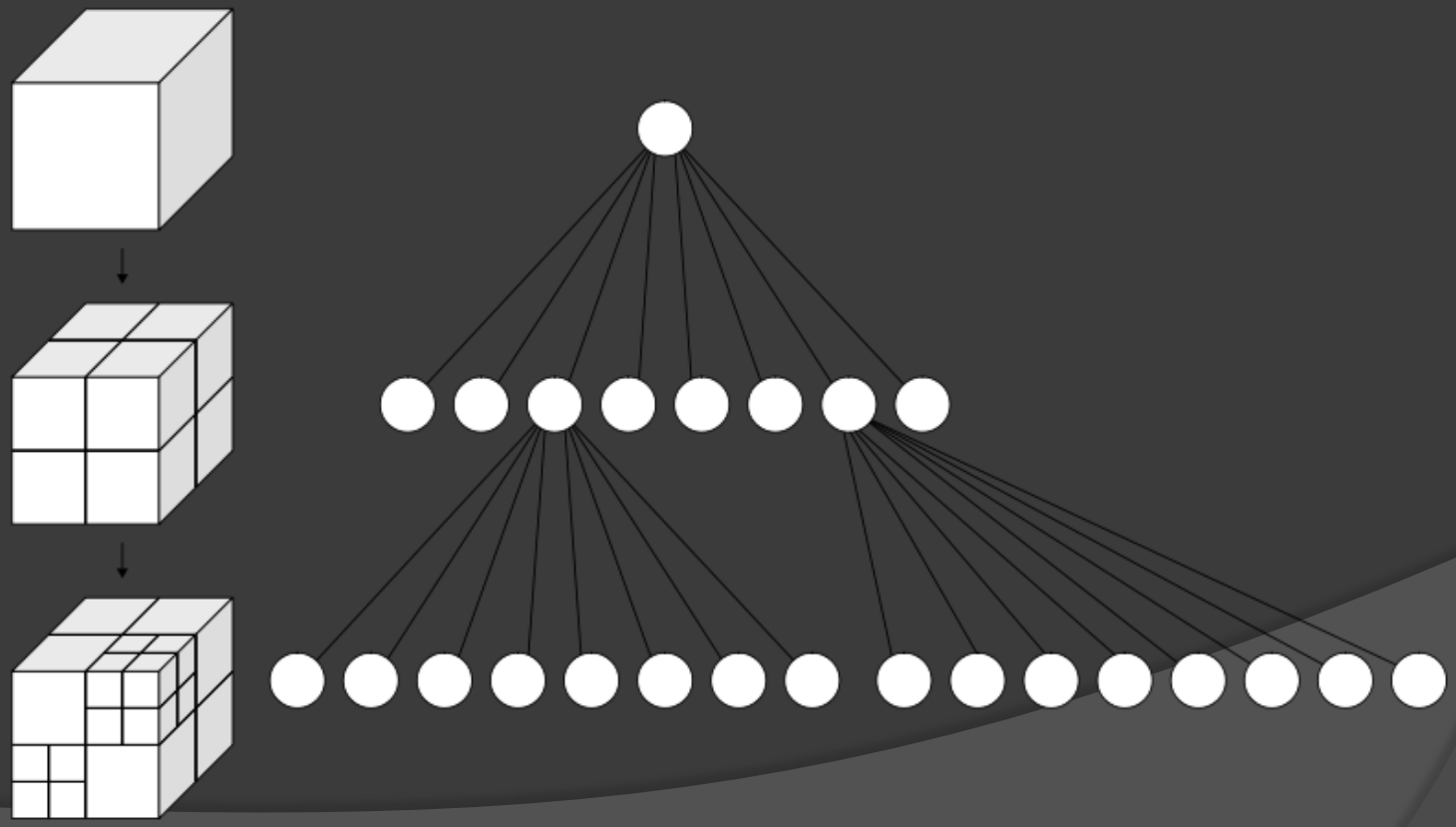
Background

- An Octree is a tree structure where every node subdivides space into eight equal pieces.



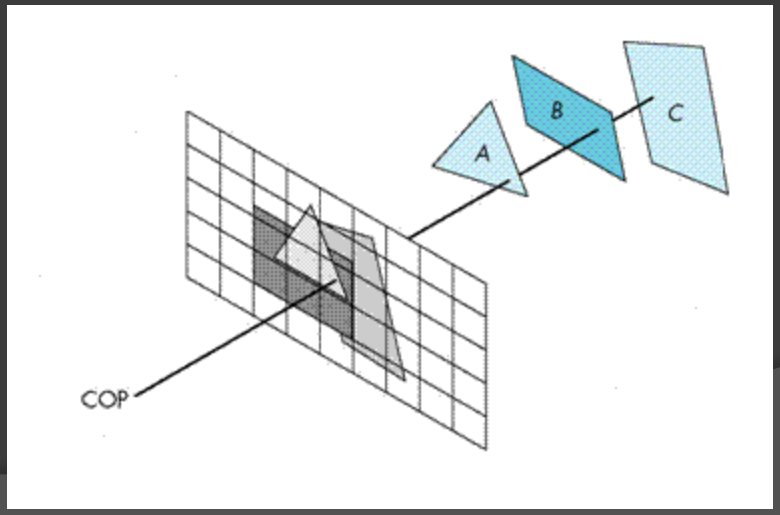
Background

- Octrees are useful because decrease the total amount of work that is required.



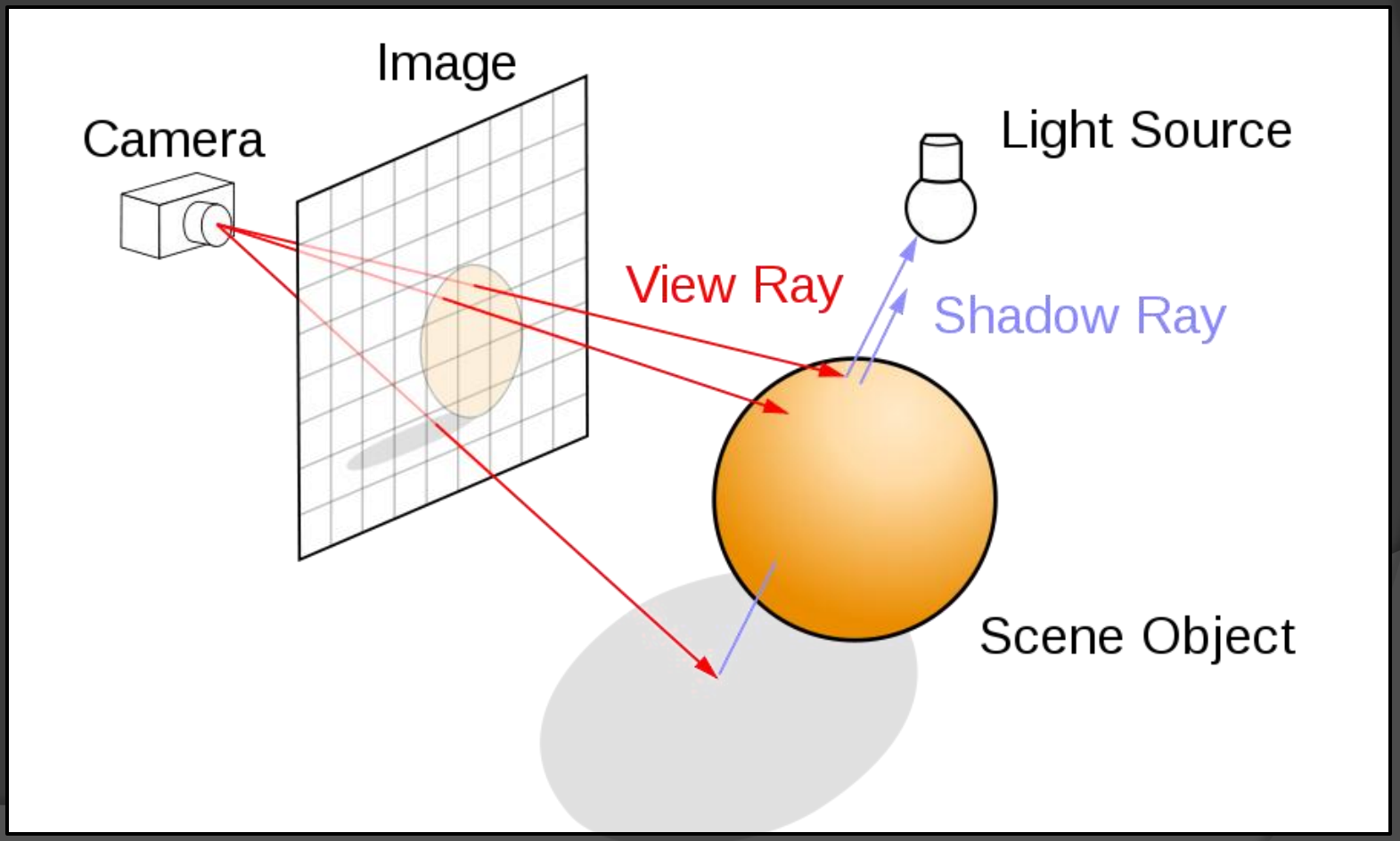
Background

- Rasterization: process of converting geometry into pixels.
- Depth-buffer test: the method of saving the distance between image or rasterization buffer and the geometry.



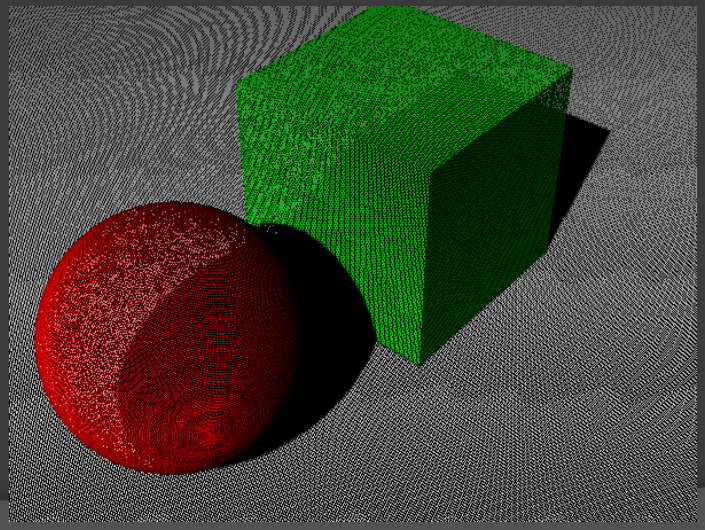
Background

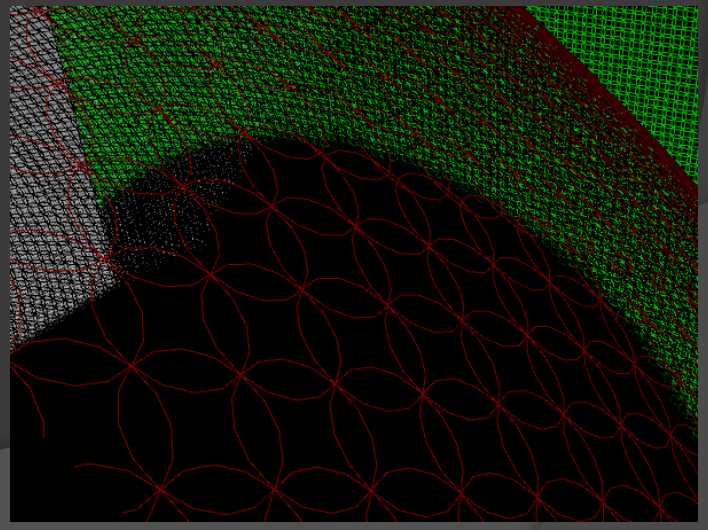
- Raytracing



Related Work

- Point Based Color Bleeding by Per H. Christianson.
  - Two Central Steps:
    - Surfel Generation
    - Rasterization



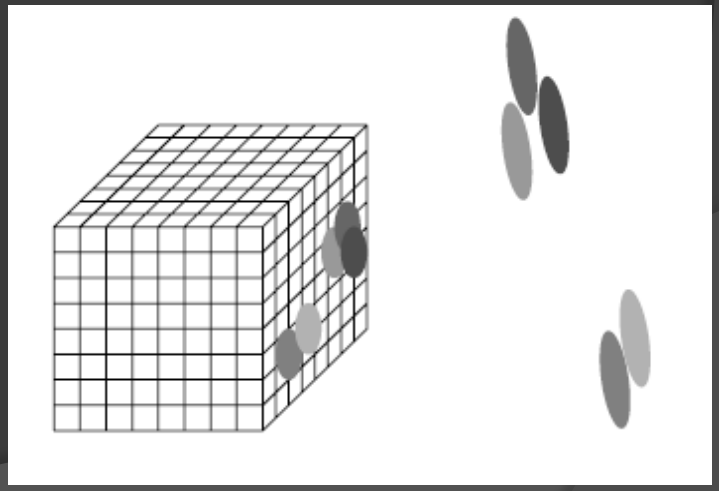


Related Work

- Surfel Generation
  - Christianson uses a REYES based approach to generate surfels. This means that each object in the scene is broken down individually and covered with surfels
  - For every Surfel direct illumination is calculated and then surfel is stored into an octree.

Related Work

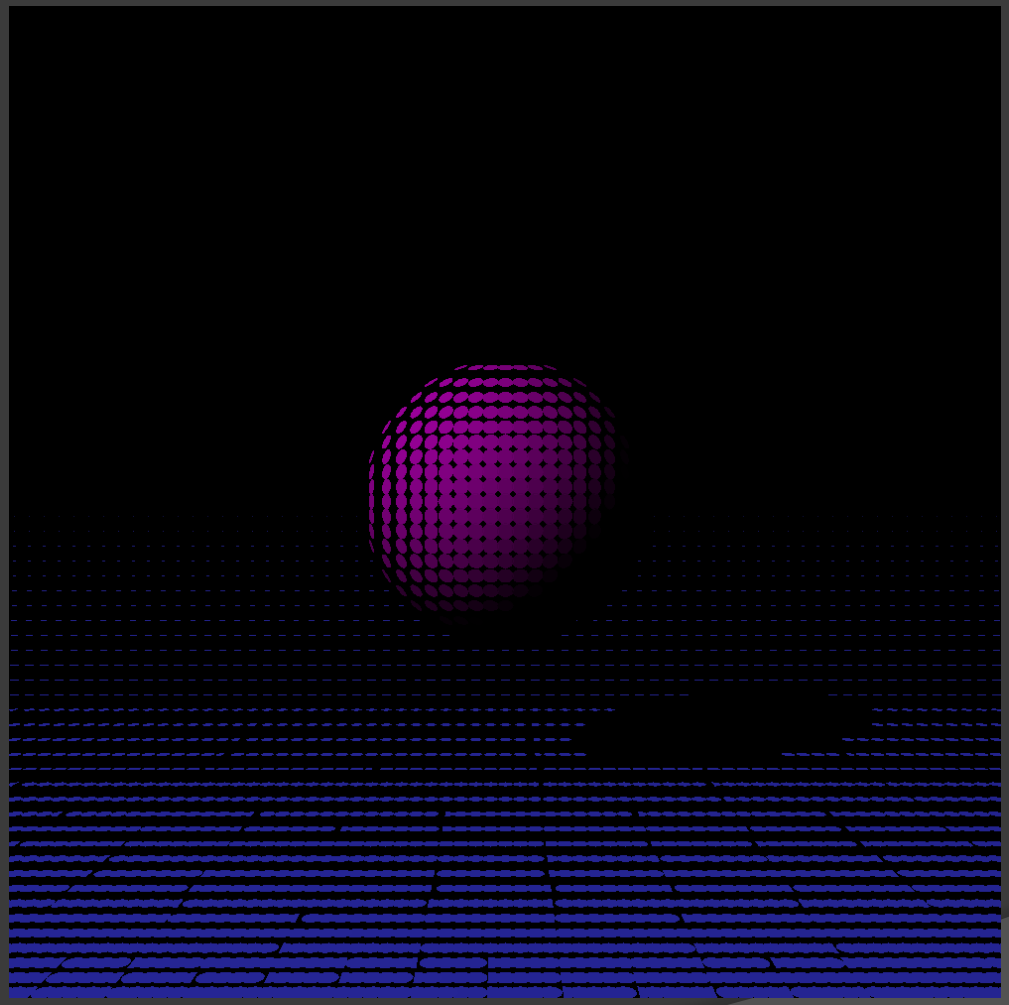
- Rasterization
  - For every point that needs to calculate global illumination a raster-cube is generated.
  - Surfels are the rasterized onto the faces of the cube.
  - Using this information global illumination is calculated.



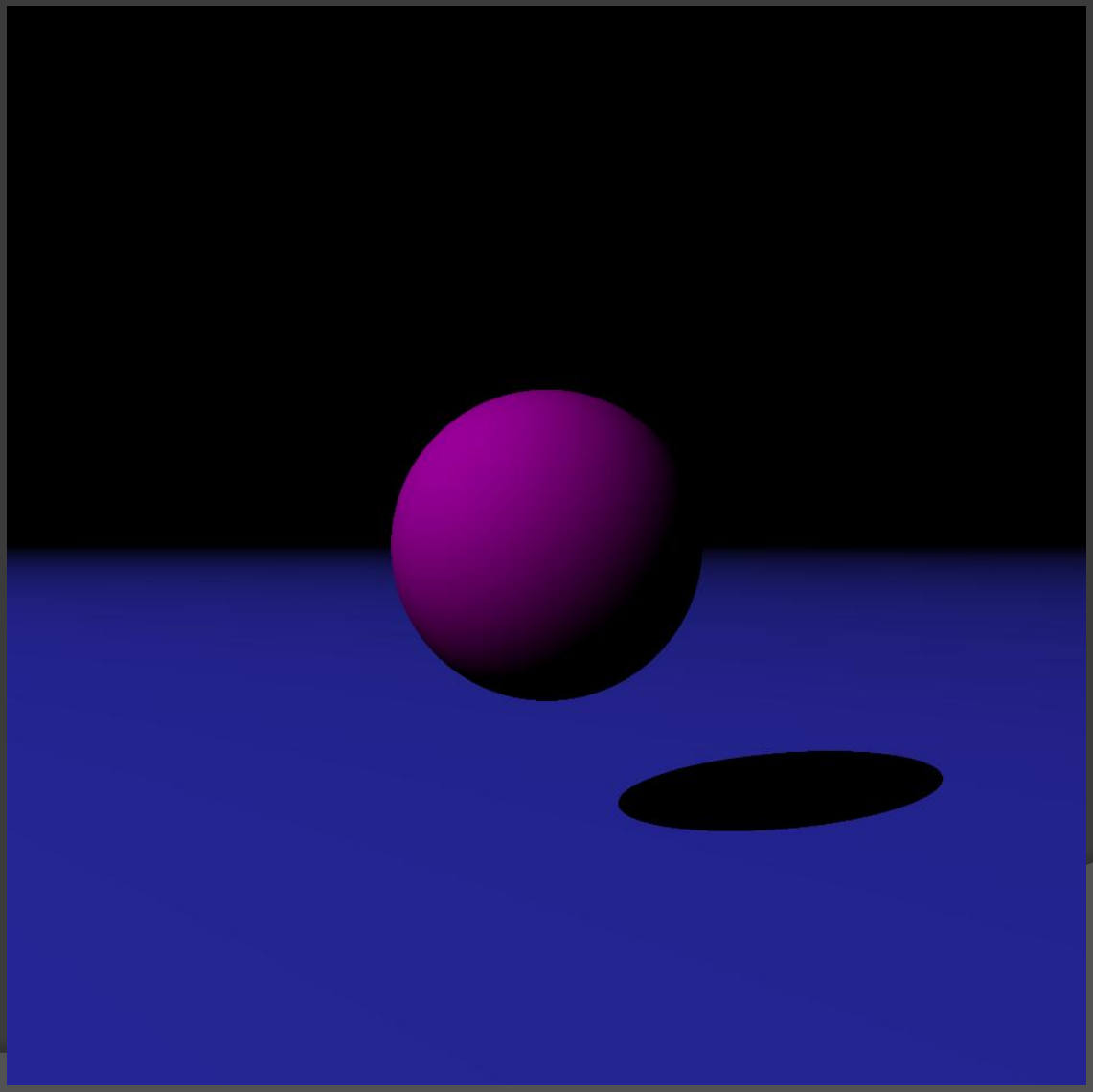
Current Implementaion

- What I did this quarter:
  - Complete C-Style Ray Tracer
    - Direct Illumination
    - Ray-Object Intersections
  - Surfel Generation
  - CPU Octree
  - CPU Surfel Raytracing
  - GPU Octree
  - GPU Surfel Raytracing

Unrealistic Surfels

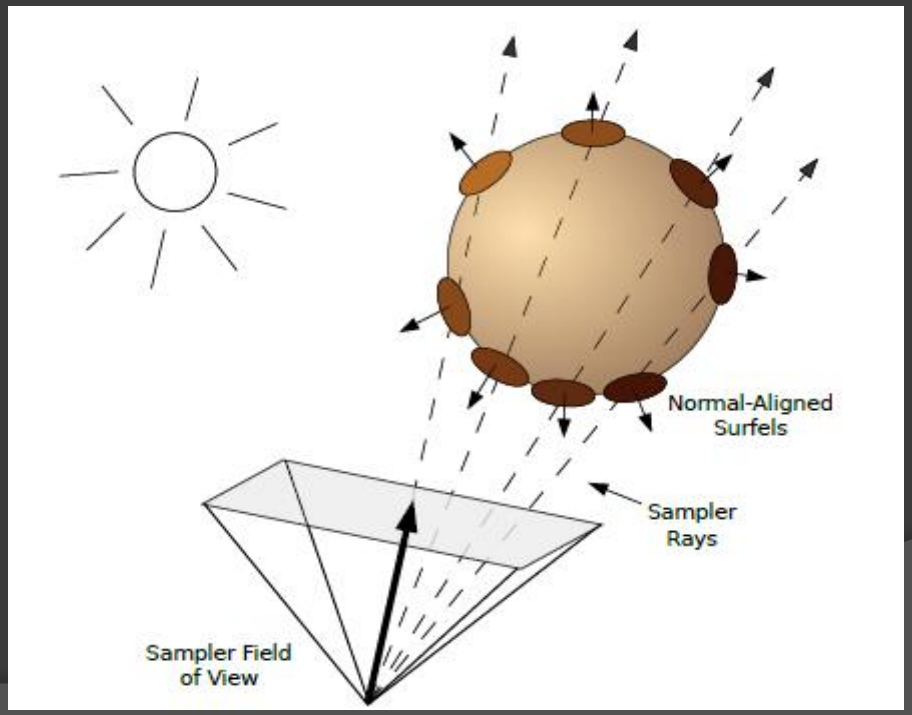


Ray Traced Surfels



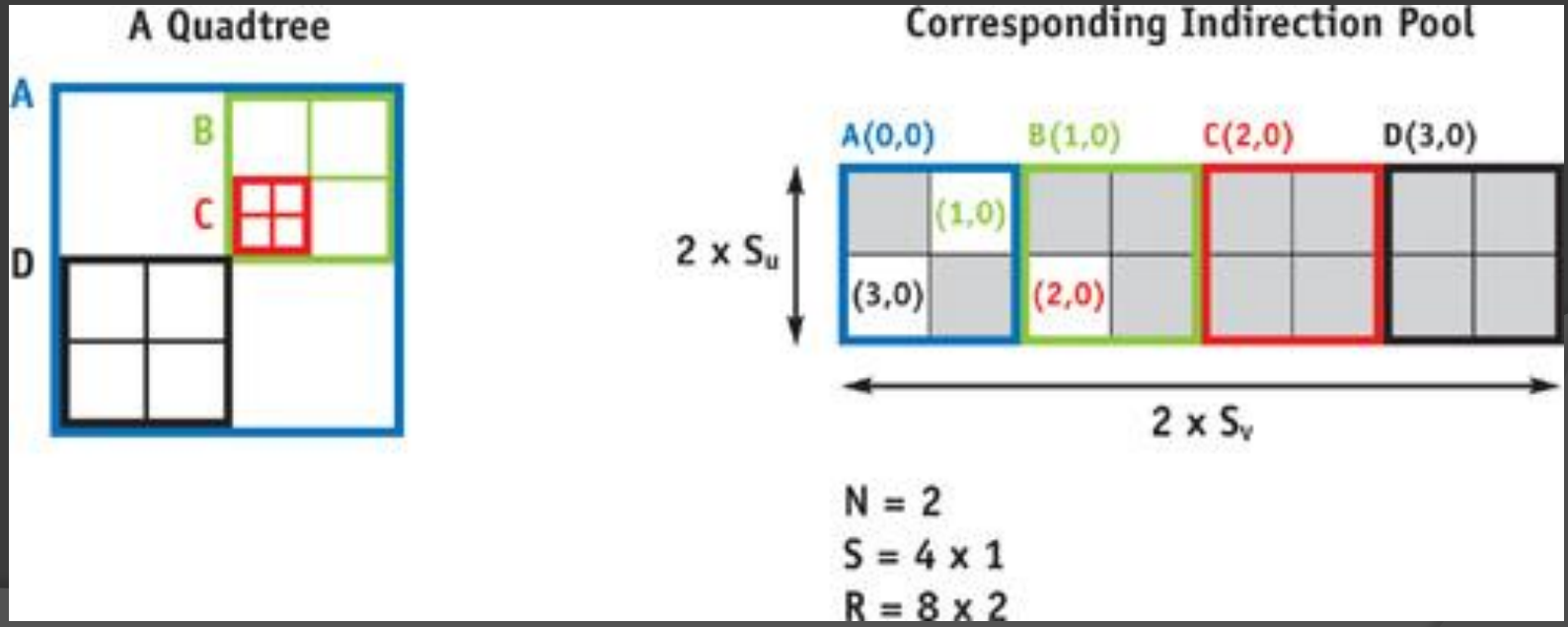
Surfel Generation

- Different from Christianson:
  - Ray Casting method
    - Enlarge View Frustum



CUDA Octree

- Normal Octrees do not work with CUDA.
- Adapt Octree to use a array structure.
- This style of Octree took 30 sec of run time



Result Times

2000x2000 Image

- CPU Non-Octree implementation:
  - The world may never know…
- CPU Pointer-Octree: 5 min 15 secs
- CPU Array-Octree: 4 min 40 secs
- CUDA non-Octree:
  - Had problems: 10 min 43 secs
- CUDA Array-Octree: 13 secs

Future Work

- Rasterization Step
- Spherical Hermonics Calculations
- Octree for Ray-Object Intersections
- CUDA for Ray-Object Intersections
- CUDA Optimizations for Surfel drawings
- Implementation on the new Kepler Nvidia Hardware

Questions?