# CS 563 Advanced Topics in Computer Graphics
## *Flexible PBR on Mobile Devices*

by Suman Nadella

# Outline

- PBR
- Overview
- Sampling
- Recursive Grids
- Memory consumption
- Rendering Algorithm
- Flexible Rendering
- Shadows
- Results
- Conclusion

# Already Discussed ...

- Point Based Rendering
  - Separates Geometry
  - Low memory Requirements
- QSplat
  - Hierarchical bounding spheres
  - LoD Control
- Surfels, surface splatting, network transmission etc ...

# Overview

- Diverse Display Devices
  - PDAs
  - Mobile Phones
- Limited Memory
- Limited CPU
- No Floating Point support
- No Graphics Hardware
- Small Display ( 240 x 320 )

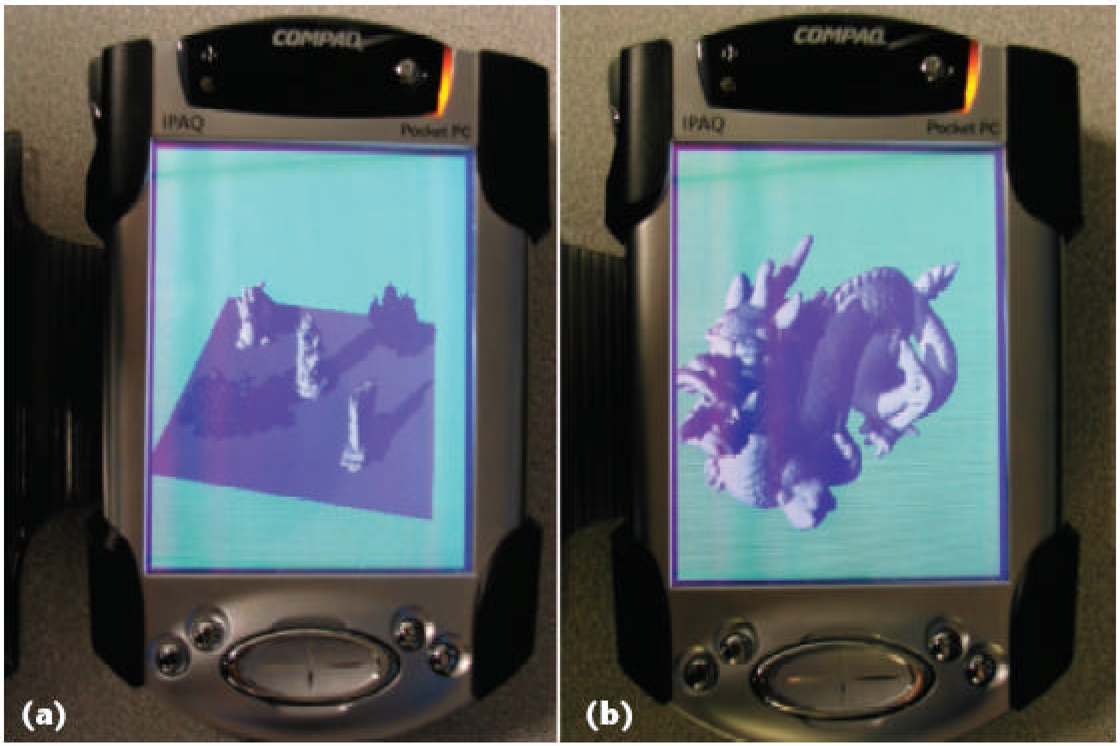
# Overview – Contd.

- Complex Scenes
  - 3D Scanning
- Millions of Polygons ~ Few Thousand Pixels
- Standard Rasterization on Polygon Geometry
  - Waste!
- Hierarchical Point Based representation
  - Used already for Data Storage and Rendering
  - More Flexible
  - Locally Adaptive Progressive Rendering
  - Explicit Storage of intermediate Attributes

## Point Sampling

- Pfister et al Octree Representation of Surfels
- Botsch et al – code point positions
  - Split all marked cells to some level
  - For each non leaf store 8 bit childhood code
  - Empty cells are free
  - Very efficient
- Generalize Botsch approach
- Extend by storing intermediate attribute samples
- Combines
  - Hierarchical rendering of QSplat
  - Compactness of Botsch et al

# Screenshot




1 Displaying models using our point-based rendering scheme on a 200-MHz iPAQ. (a) Structure used for hierarchical point-based rendering of a 4.7-Mbyte polygon model at 2.1 frames per second (fps), sampled at 1.3-Mbyte points. The multilevel approach restricts the number of points rendered depending on the view. (b) The dragon model at 2.3 fps.

# Framework

- Sampling Strategy
  - For each cell, sample geometry at the point of object closest to the center of the cell
  - If Object is not present, flag as empty
  - Thus we can sample as long as ...
  - IntersectAxisAlignedBox
    - Returns true if object intersects or is contained in the given axis aligned box
  - GetSampleAt
    - Returns a sample and its attributes for a given input point (cell center) and geometry primitive.

## Framework – Contd.

- Sample position is coded implicitly in the hierarchy
- Normal and Material Indices
- 16 bit code
  - 13 bit quantized normal index
  - Indexing 8 materials
- More materials/colors – more bits
  - 32 to 48 bits
- Recursive Grid Structure
  - Intermediate Sample Attributes
  - Normal/Color for interior nodes

## Recursive Grids

- First used in Ray Tracing
- Flexible and optimized traversal
- ? grids
  - subcells are $?^3$
  - ? subcells per dimension
- Octree – simplest grid
  - ? = 2 , 2x2x2
- Each cell is subdivided into eight subcells
- Tri-grid
  - ? = 3 , 3x3x3

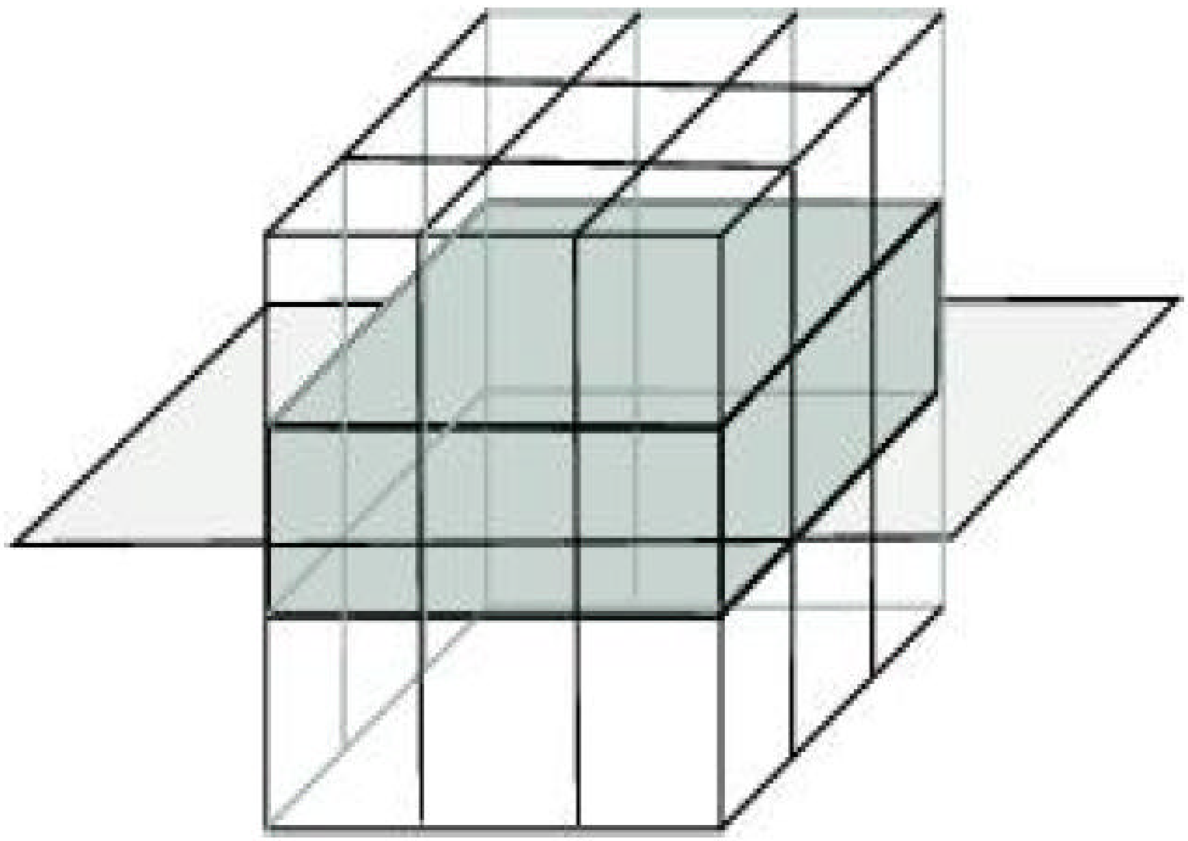
# Recursive Grids – Contd.

- Position of sample aligned with cell center
- So need not be stored
- Extend algorithm for Octree (Botsh) to ? grids
- 27 bits to encode position
- Set the bit if sub-cell is non-empty

# Memory Consumption

- a be number of non-empty sub-cells per cell
- a? be the average of a over all cells
- Tri-grid with subdivision depth >= 5
  - a? approximates to 9
  - Independent of model type
- d = log a? / log ?
- For a tri-grid, d measures to be 2
- Intuitively, d is related to dimension of models

# Illustration



# Memory – Contd.

- n = number of samples in the best model
  - Same as number of leaf cells
- Take both position and attributes into account
- m = maximum depth of ? grid
- Ni = number of cells at depth I
- Nm = n and Ni = a? Ni-1
- Intermediate cells is sum over all levels except last

$$N = \sum_{i=0}^{m-1} N_i = \frac{n-1}{\rho^d - 1}$$

# Memory – Contd.

- Cost of structure by sample
- Divide by n
- Counting ?[3] bits per cell

$$S_c = \frac{\rho^3}{\rho^d - 1}$$

- Size increases with ?
- Octree is optimal for storage
  - If only position coding is considered

## Intermediate Samples

- If sample attributes are also considered
- Number of intermediate samples = number of intermediate cells
- Size in bits of a sample attribute – sigma
- Memory cost, per leaf sample is

$$S_s = \frac{\sigma}{\rho^d - 1}$$

- Total cost

$$S = \frac{\sigma + \rho^3}{\rho^d - 1}$$

# Variations in Consumption

| $\sigma$ | $\rho = 2$ | $\rho = 3$ | $\rho = 4$ | $\rho = 5$ |
|---|---|---|---|---|
| 0 | **2.66** | 3.38 | 4.26 | 5.21 |
| 4 | 4 | **3.88** | 4.53 | 5.375 |
| 8 | 5.33 | **4.38** | 4.8 | 5.54 |
| 12 | 6.67 | **4.88** | 5.06 | 5.71 |
| 16 | 8 | 5.38 | **5.33** | 5.88 |
| 32 | 13.33 | 7.38 | **6.4** | 6.54 |
| 48 | 18.67 | 9.38 | 7.47 | **7.21** |

- Tri-grid has lowest consumption for 4-16 bits
- Higher bits – good for 4 or 5 grids
- Not efficient for rendering

# Rendering

- Rendering needs a 4x4 multiplication
- Projection to screen coordinates
- For each vertex of the model
- Min – 16 multiplications and 12 additions
- Polygons not suitable for mobile devices
- Structured hierarchies
- Generalized rendering of Octrees to ? grids

## Basic Algorithm

- Given a ? grid, project the center
- Standard projection in homogenous coords
- Precompute displacement vector table
- $?^3$ vectors, corresponding to sub-cell centers
- Linear projections
- So, sub-cell projection computed from parent center and displacement vector
- Computed for each level and each modification of viewing position

# Basic Algorithm – Contd.

- Center of a cell in ? grid takes three additions from center of the parent cell
- Final projection – dehomogenize
- Two divisions
- Displacement vector table $d_{i,j,k}$ Precomputed
- Displacements giving first level from root –

$$\vec{d}^{(1)}_{i,j,k} = \frac{i}{\rho}\vec{e_i} + \frac{j}{\rho}\vec{e_j} + \frac{k}{\rho}\vec{e_k}$$

$i, j, k \in [-\frac{\rho-1}{2}; \frac{\rho-1}{2}]$ $\quad \vec{e_i}, \vec{e_j}, \vec{e_k}$

projected unit basis vectors

## Basic Algorithm – Contd.

- Subsequent levels
- Incremental computation
- One multiplication per vector

$$\vec{d}^{(n)}_{i,j,k} = \frac{1}{\rho}\vec{d}^{(n-1)}_{i,j,k}$$

- For each sub-cell i,j,k of a cell c, projected center is computed with three additions

$$\vec{c}^{(n)}_{i,j,k} = \vec{d}^{(n)}_{i,j,k} + \vec{c}^{(n-1)}$$

# Recursive Rendering

- Algorithm

```
Render (cell, center, level)
    if cell is a leaf
        for each subcell
            if sampled
                compute position
                Draw Sample
    else
        for each subcell
            if exists
                compute subcenter
                Render (subcell, subcenter, level+1)
```

# Flexible Rendering

- Till now always render leafs
- Waste if projected size of intermediate cells is less than a pixel
  - When zooming out of the object
- Can reconstruct attributes by averaging
- More expensive than rendering itself
  - Averaged Quantized normals (Botsch)
- Method for rendering intermediate nodes

# Flexible Rendering – Contd

- Conservative approximation of cell projection
- Screen space bounding rectangle of the cell
- Displacement table
- Bounds ($min_x$,$min_y$,$max_x$,$max_y$)
- $min_{z/w}$ – min homogenous depth of the box
- For each level, if extent of box is less than a pixel, draw intermediate sample
- Splats to represent samples
- $d_x = (max_x - min_x)$ = extent in x , s is splat size
- Two tests to determine which level to stop…

# Tests to check level

$$\frac{dx}{w} \leq s, w > 0 \qquad dx \leq w * s, w > 0$$

- Efficient frustum culling test using same info
- Bounds of projected cell against screen bounds
- If intermediate cell is outside, ignore it and its children
- If not, precompute the extent S as

$$S = max_{x,y}(max - min)$$

avoiding all min-max computations

# Shadows

- Efficient shadow map computation
- Can use larger splats – less details needed
- If standard algorithm,
  - For each pixel, transform to light source space
  - 3x3 matrix – 9 mul, 6 add, one shadow map test
  - iPAQ screen – 690,000 mul, 540,000 add
- Avoid matrix multiplications , two passes
  - Render to a depth map
  - Render the scene

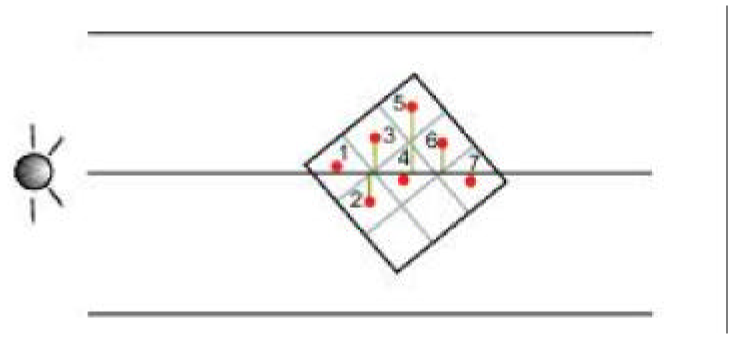
# Shadows – Contd.

- Render to a depth map using projection matrix of the light source and displacement vectors
- Each sample in light space uses the incremental technique
- Avoids matrix computations to transform points into light space
- Render scene computing positions for light source proj matrix and camera proj matrix
- Perform depth map test for each sample

# Shadows – Single Pass

- Single pass method for directional light sources
- Given light direction, attribute strict order in rendering of hierarchical structure
- Ordering of sub-cells to render back to front
- Precomputed once for the 27 cells for a given light source position
- Project subcell centers onto light source direction
- Lit samples are rendered first, and shadows later
- Compute shadow map and view from camera and perform depth test as we render
- So avoiding first pass of earlier method

# Shadows - Illustration




- Can be used in general for any ? grid
- tri-grid is justified for our use

| Model | Points | splats | shadows | One-Pass |
|---|---|---|---|---|
| Buddha (4) | 5.49 (4.15) | 3.47 | 2.65 | 2.99 |
| Buddha (5) | 0.91 (0.63) | 3.33 | 2.46 | 2.85 |
| Blade (4) | 4.71 (3.30) | 2.63 | 1.99 | 2.32 |
| Blade (5) | 0.67 (0.47) | 2.40 | 1.89 | 2.21 |
| Big Scene | 0.62 (0.30) | 2.38 | 1.83 | 2.11 |

# Rendering Cost

- number of cells and number of ops per cell
- 3 additions per subcell 3?$^d$
- Shift per subcell - ?$^3$
- Intermediate cells per sample – 1 / (?$^d$ -1)
- Thus number of operations

$$T = \frac{3\rho^d + \rho^3}{\rho^d - 1}$$

- Only basic rendering – point samples
- No shadows, splats, frustum culling

# Comparison of Costs

| $\rho$ | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| basic (3) | 6.6 | 6.7 | 7.5 | 8.4 |
| shadows (6) | 10.7 | 10.1 | 10.6 | 11.5 |

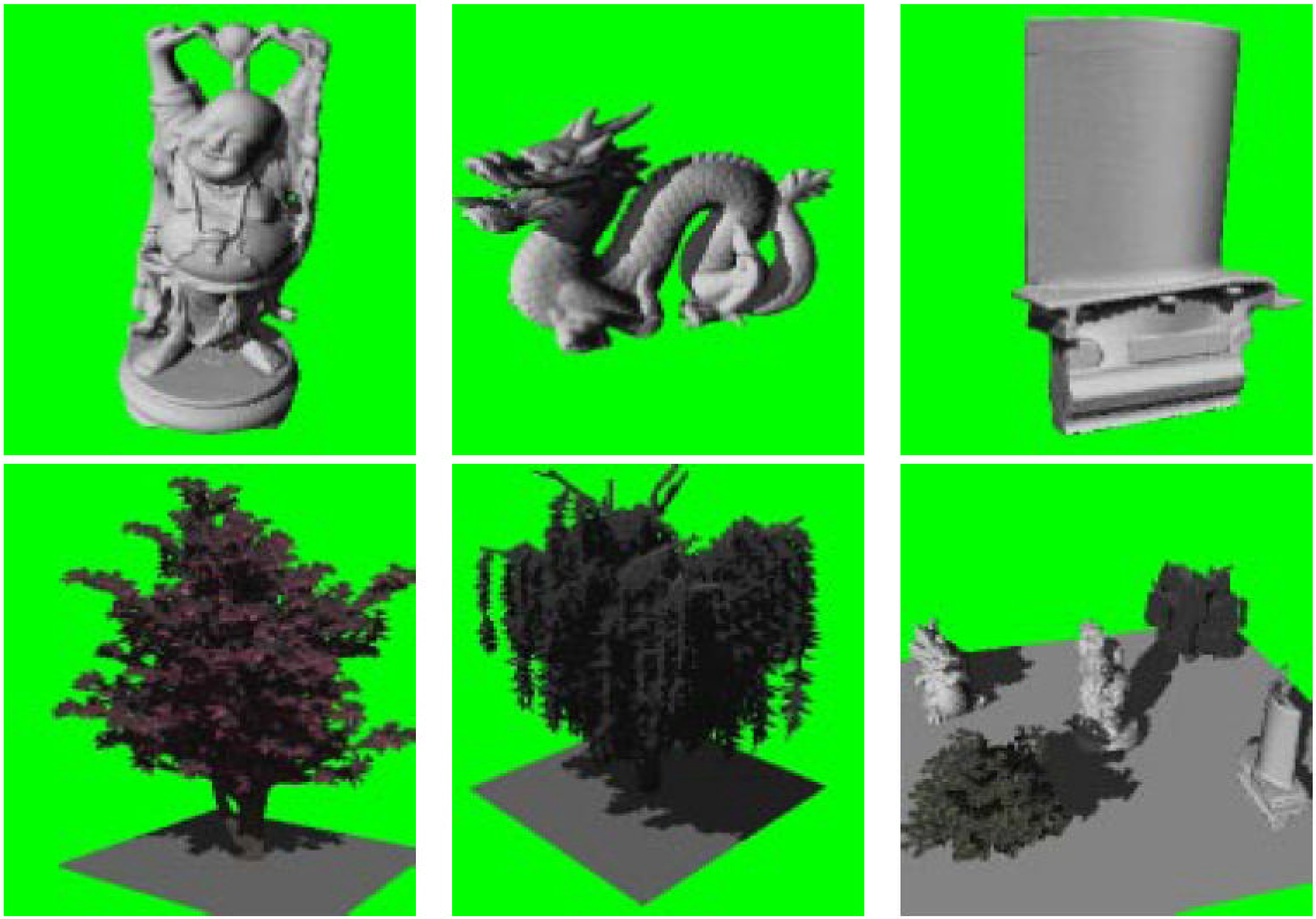
- Tri-grid is as efficient as Octree
- Better than a 4,5 grid
- So best bet when taking considering both rendering and storage
- Larger grids can cause jumps - switching levels
- Precomputation of disp vectors - negligible

# Pre-rendering materials

- Shading is precomputed for each possible normal direction
- $2^{13}$ = 8k directions
- Shading per material – table of evaluations of material properties for each lighting angle
- Simply use a lookup table
- Precision of quantization values – level 5 for 16 bits color displays

| level | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| error | 0.017 | 0.004 | **0.001** | 2.6 e-4 | 6.4 e-5 |

# Results



# Implementation

- Tri-grids computed on workstation
- Transferred to iPAQ using a Flash Card
- 200 MHz processor
- 64 Mbytes main memory
- 8-9 Mbytes available for datastructure
- 16 bit attribute samples
  - 13 bit normals
  - 3 bit material index

# Implementation Issues

- No Floating point support
- Division can be more expensive than lookup
- Implemented ...
- Fixed Point Arithmetic
- Approximation of inverse upto a given precision using lookup table
- 32 bit fixed point numbers
- Shift on numbers rather than global lookup
- Based on expected precision

# Statistics

| Model | polygons | Samples 3 | Samples 4 | Samples 5 | $\alpha$ 3 | $\alpha$ 4 | $\alpha$ 5 |
|---|---|---|---|---|---|---|---|
| Bunny | 69 k | 26 k | 230 k | 2.1 M | 9.36 | 9.03 | 9.00 |
| Dragon | 870 k | 18 k | 170 k | 1.5 M | 9.75 | 9.19 | 9.04 |
| Buddha | 1.08 M | 14 k | 130 k | 1.2 M | 10.3 | 9.45 | 9.07 |
| Blade | 1.76 M | 17 k | 180 k | 1.7 M | 13.19 | 10.77 | 9.26 |
| Arbre | 540 k | 40 k | 370 k | 3.2 M | 11.45 | 9.18 | 8.63 |
| Saule | 420 k | 45 k | 430 k | 3.4 M | 13.62 | 9.62 | 7.84 |
| Big Scene | 4.67 M | 134 k | 1.28 M | 11.0 M | - | - | - |

- Cubic nature of cells affect number of samples
- Alpha value – well behaved models = 9
- Other models, level 3 sampling is not fine enough
- At level 5, the value stabilizes around 9
- Tree models – edge effect due to leaves, doesn’t stay at 9

# Statistics – Contd.

| Model | wrl.gz file | File Size 3 | 4 | 5 |
|---|---|---|---|---|
| Bunny | 858 kb | 76.8 kb | 698 kb | 6.28 Mb |
| Dragon | 8.90 Mb | 54.5 kb | 506 kb | 4.60 Mb |
| Buddha | 11.0 Mb | 41.5 kb | 396 kb | 3.65 Mb |
| Blade | 14.4 Mb | 46.2 kb | 520 kb | 5.06 Mb |
| Arbre | 8.53 Mb | 114 kb | 1.1 Mb | 9.77 Mb |
| Saule | 9.31 Mb | 121 kb | 1.28Mb | 10.7 Mb |
| Big Scene | 52.14 Mb | 377 kb | 4.5 Mb | 40 Mb |

- Core data structure increases by 25% - pointers
- 8Mbytes available – can fit in all level 4 models
- If not, a combination of 3,4,5

# Results – Contd.

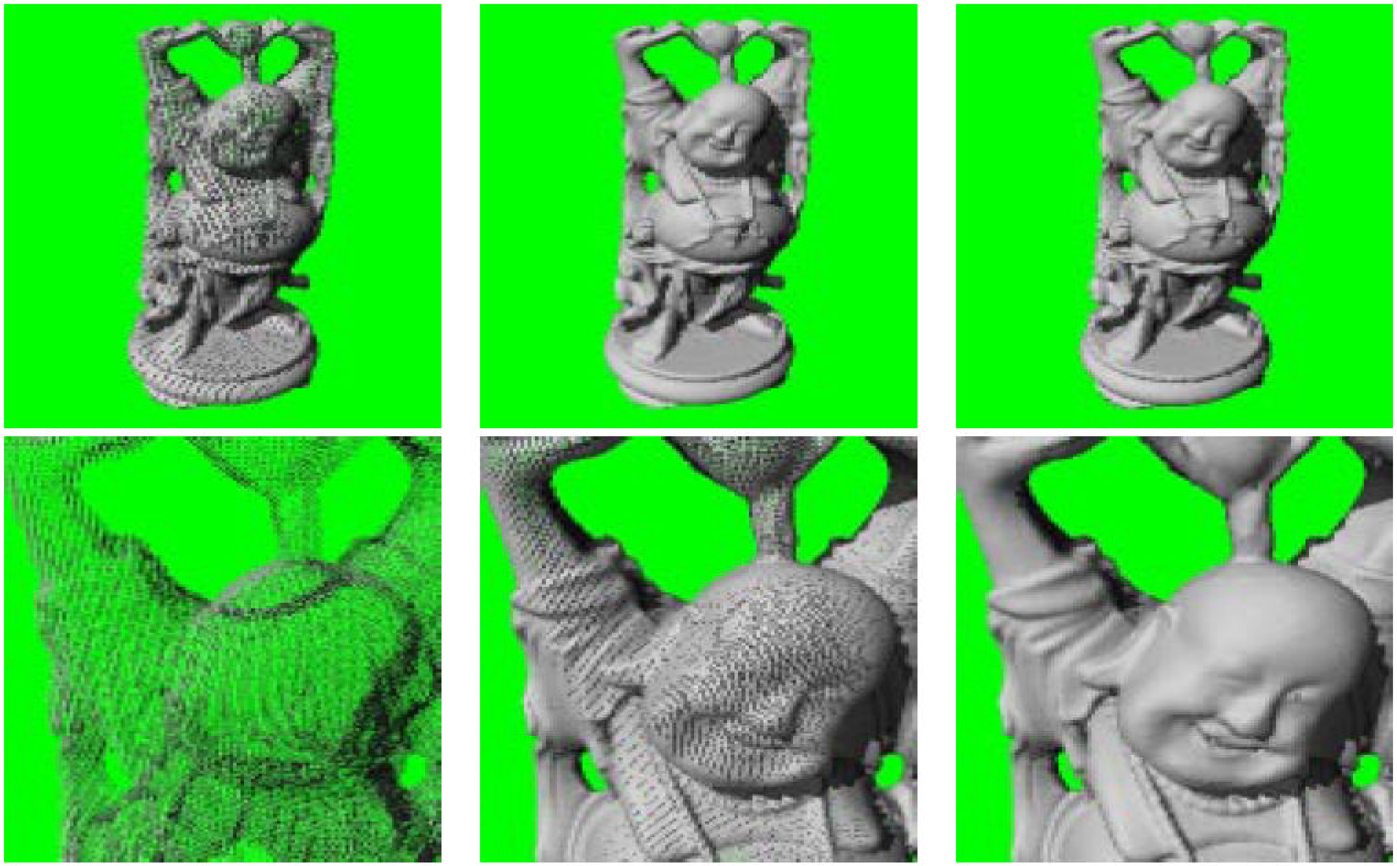
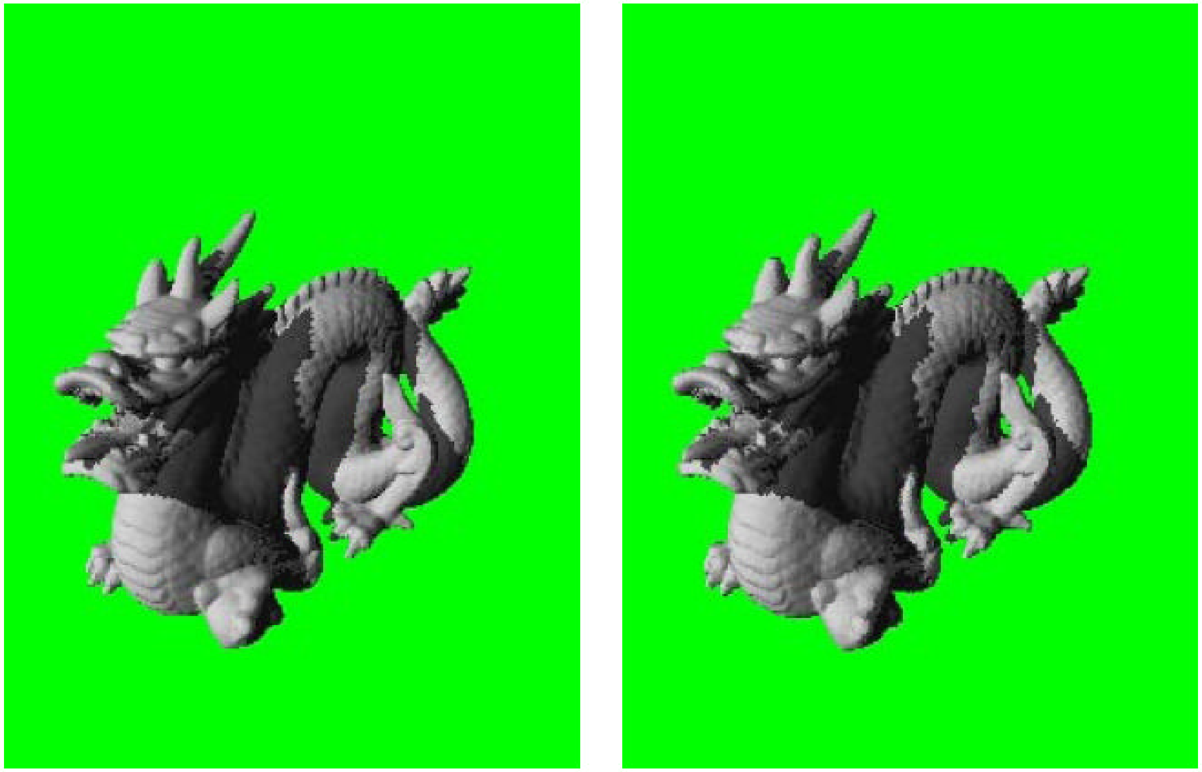


Figure 5: Above, a far view of the Buddha model shown (left) with points at level 4 (4.15 fps) (middle) with points at level 5 and (0.63 fps) (right) with splats at level 5 (2.85 fps). Below, a close view of the Buddha model shown (left) with points at level 4 (4.15 fps) (middle) with points at level 5 and (0.63 fps) (right) with splats at level 5 (1.42 fps). Undersampling problems are evident at level 4 subdivision when using points only. At level 5, the increase in frame rate is notable.

## Results – Contd.



0.55 fps    2.28 fps

Figure 6: Quality and speed of shadows. View of the Dragon model shown (left) with points at level 5 with shadows and (right) multi-level display with one-pass shadows multi-level. Notice that shadows generated with multi-level rendering are practically indistinguishable from shadows generated with full point resolution.

# Video

Results Video

# References


- Florent Duguet George Drettakis, Flexible Point-Based Rendering on Mobile Devices , *INRIA tech report RR-4833*

  http://www-sop.inria.fr/reves/publications/data/2003/DD03/RR-4833.pdf
- Florent Duguet George Drettakis, Flexible Point-Based Rendering on Mobile Devices , *IEEE Computer Graphics and Applications number 4 volume 24 July-August 2004*

  http://csdl.computer.org/comp/mags/cg/2004/04/g4057abs.htm
- M. Botsch, A. Wiratanaya, and L. Kobbelt, Efficient High-Quality Rendering of Point-Sampled Geometry,*Rendering Techniques 2002*, *Eurographics*

  http://portal.acm.org/citation.cfm?id=581904
- Pfister, et. al., Surfels: Surface Elements as Rendering Primitives, *Siggraph 2000*

  www.merl.com/people/pfister/pubs/sig2000.pdf
- Florent Duguet Homepage

  http://www-sop.inria.fr/reves/Florent.Duguet/

# Thank You

Questions/Comments/Suggestions